

编程狂人

Programming Madman

NO. 40



关于推酷

推酷是专注于IT圈的个性化阅读社区。我们利用智能算法,从海量文章资讯中挖掘出高质量的内容,并通过分析用户的阅读偏好,准实时推荐给你最感兴趣的内容。我们推荐的内容包含科技、创业、设计、技术、营销等内容,满足你日常的专业阅读需要。我们针对IT人还做了个活动频道,它聚合了IT圈最新最全的线上线下活动,使IT人能更方便地找到感兴趣的活动信息。

关于周刊

《编程狂人》是献给广大程序员们的技术周刊。我们利用技术挖掘出那些高质量的文章,并通过人工加以筛选出来。每期的周刊一般会在周二的某个时间点发布,敬请关注阅读。

本期为精简版 周刊完整版链接:

<http://www.tuicool.com/mags/5404594cd91b14339d011866>

欢迎下载推酷客户端体验更多阅读乐趣



版权说明

本刊只用于行业间学习与交流署名文章及插图版权归原作者享有

目录

- 01.快速入门：十分钟学会Python
- 02.关于Python编程的一些问答
- 03.JavaScript构建（编译）系统大比拼
- 04.为什么Android的图片质量会比iPhone的差？
- 05.一次app抓包引发的Android分析记录
- 06.Whitepages的架构变迁：从Ruby到响应性更好的Scala和Akka
- 07.美团云的技术演变：先把云主机做稳定了再说别的
- 08.基于Storm的Nginx log监控系统
- 09.专访张路斌：从HTML 5到Unity的游戏开发之路
- 10.Lisp魔咒：对Lisp的非技术性吐槽

快速入门：十分钟学会Python

作者：Stavros Korokithakis

初试牛刀

假设你希望学习Python这门语言，却苦于找不到一个简短而全面的入门教程。那么本教程将花费十分钟的时间带你走入Python的大门。本文的内容介于教程(Tutorial)和速查手册(CheatSheet)之间，因此只会包含一些基本概念。很显然，如果你希望真正学好一门语言，你还是需要亲自动手实践的。在此，我会假定你已经有了了一定的编程基础，因此我会跳过大部分非Python语言的相关内容。本文将高亮显示重要的关键字，以便你可以很容易看到它们。另外需要注意的是，由于本教程篇幅有限，有很多内容我会直接使用代码来说明加以少许注释。

Python的语言特性

Python是一门具有强类型(即变量类型是强制要求的)、动态性、隐式类型(不需要做变量声明)、大小写敏感(var和VAR代表了不同的变量)以及面向对象(一切皆为对象)等特点的编程语言。

获取帮助

你可以很容易的通过Python解释器获取帮助。如果你想知道一个对象(object)是如何工作的，那么你所需要做的就是调用 `help(<object>)`！另外还有一些有用的方法，`dir()`会显示该对象的所有方法，还有 `<object>.__doc__`会显示其文档：


```
1 >>> help(5)
2 Help on int object:
3 (etc etc)
4
5 >>> dir(5)
6 ['__abs__', '__add__', ...]
7
8 >>> abs.__doc__
9 'abs(number) -> number
10
11 Return the absolute value of the argument.'
```

语法

Python中没有强制的语句终止字符，且代码块是通过缩进来指示的。缩进表示一个代码块的开始，逆缩进则表示一个代码块的结束。声明以冒号(:)字符结束，并且开启一个缩进级别。单行注释以井号字符(#)开头，多行注释则以多行字符串的形式出现。赋值（事实上是将对象绑定到名字）通过等号(“=”)实现，双等号(“==”)用于相等判断，“+=”和“-=”用于增加/减少运算(由符号右边的值确定增加/减少的值)。这适用于许多数据类型，包括字符串。你也可以在一行上使用多个变量。例如：

```
1 >>> myvar = 3
2 >>> myvar += 2
3 >>> myvar
4 5
5 >>> myvar -= 1
6 >>> myvar
7 4
8 """This is a multiline comment.
9 The following lines concatenate the two strings."""
10 >>> mystring = "Hello"
11 >>> mystring += " world."
12 >>> print mystring
13 Hello world.
14 # This swaps the variables in one line(!).
15 # It doesn't violate strong typing because values aren't
16 # actually being assigned, but new objects are bound to
17 # the old names.
18 >>> myvar, mystring = mystring, myvar
```

数据类型

Python具有列表（list）、元组（tuple）和字典（dictionaries）三种基本的数据结构，而集合(sets)则包含在集合库 中(但从Python2.5版本开始正式成为Python内建类型)。列表的特点跟一维数组类似（当然你也可以创建类似多维数组的“列表的列表”），字典 则是具有关联关系的数组（通常也叫做哈希表），而元组则是不可变的一维数组（Python中“数组”可以包含任何类型的元素，这样你就可以使用混合元素， 例如整数、字符串或是嵌套包含列表、字典或元组）。数组中第一个元素索引值(下标)为0，使用负数索引值能够从后向前访问数组元素，-1表示最后一个元 素。数组元素还能指向函数。来看下面的用法：

```
1 >>> sample = [1, ["another", "list"], ("a", "tuple")]
2 >>> mylist = ["List item 1", 2, 3.14]
3 >>> mylist[0] = "List item 1 again" # We're changing the item.
4 >>> mylist[-1] = 3.21 # Here, we refer to the last item.
5 >>> mydict = {"Key 1": "Value 1", 2: 3, "pi": 3.14}
6 >>> mydict["pi"] = 3.15 # This is how you change dictionary values.
7 >>> mytuple = (1, 2, 3)
8 >>> myfunction = len
9 >>> print myfunction(mylist)
10 3
```

你可以使用:运算符访问数组中的某一段，如果:左边为空则表示从第一个元素开始，同理:右边为空则表示到最后一个元素结束。负数索引则表示从后向前数的位置（-1是最后一个项目），例如：

```
1 >>> mylist = ["List item 1", 2, 3.14]
2 >>> print mylist[:]
3 ['List item 1', 2, 3.1400000000000001]
4 >>> print mylist[0:2]
5 ['List item 1', 2]
6 >>> print mylist[-3:-1]
7 ['List item 1', 2]
8 >>> print mylist[1:]
9 [2, 3.14]
10 # Adding a third parameter, "step" will have Python step in
11 # N item increments, rather than 1.
12 # E.g., this will return the first item, then go to the third and
13 # return that (so, items 0 and 2 in 0-indexing).
14 >>> print mylist[::2]
15 ['List item 1', 3.14]
```

字符串

Python中的字符串使用单引号(')或是双引号(")来进行标示, 并且你还能在通过某一种标示的字符串中使用另外一种标示符(例如 "He said 'hello'.")。而多行字符串可以通过三个连续的单引号('')或是双引号("")来进行标示。Python可以通过u"This is a unicode string"这样的语法使用Unicode字符串。如果想通过变量来填充字符串, 那么可以使用取模运算符(%)和一个元组。使用方式是在目标字符串中从左至右使用%s来指代变量的位置, 或者使用字典来代替, 示例如下:

```
1 >>> print "Name: %s\  
2 Number: %s\  
3 String: %s" % (myclass.name, 3, 3 * "-")  
4 Name: Poromenos  
5 Number: 3  
6 String: ---  
7  
8 strString = """This is  
9 a multiline  
10 string."""  
11  
12 # WARNING: Watch out for the trailing s in "%(key)s".  
13 >>> print "This %(verb)s a %(noun)s." % {"noun": "test", "verb": "is"}  
14 This is a test.
```

流程控制

Python中可以使用if、for和while来实现流程控制。Python中并没有select, 取而代之使用if来实现。使用for来枚举列表中的元素。如果希望生成一个由数字组成的列表, 则可以使用range(<number>)函数。以下是这些声明的语法示例:


```

1 rangelist = range(10)
2 >>> print rangelist
3 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4 for number in rangelist:
5     # Check if number is one of
6     # the numbers in the tuple.
7     if number in (3, 4, 7, 9):
8         # "Break" terminates a for without
9         # executing the "else" clause.
10        break
11    else:
12        # "Continue" starts the next iteration
13        # of the loop. It's rather useless here,
14        # as it's the last statement of the loop.
15        continue
16 else:
17     # The "else" clause is optional and is
18     # executed only if the loop didn't "break".
19     pass # Do nothing
20
21 if rangelist[1] == 2:
22     print "The second item (lists are 0-based) is 2"
23 elif rangelist[1] == 3:
24     print "The second item (lists are 0-based) is 3"
25 else:
26     print "Dunno"
27
28 while rangelist[1] == 1:
29     pass

```

函数

函数通过“def”关键字进行声明。可选参数以集合的方式出现在函数声明中并紧跟着必选参数，可选参数可以在函数声明中被赋予一个默认值。已命名的参数需要赋值。函数可以返回一个元组（使用元组拆包可以有效返回多个值）。Lambda函数是由一个单独的语句组成的特殊函数，参数通过引用进行传递，但对于不可变类型(例如元组，整数，字符串等)则不能够被改变。这是因为只传递了该变量的内存地址，并且只有丢弃了旧的对象后，变量才能绑定一个对象，所以不可变类型是被替换而不是改变（译者注：虽然Python传递的参数形式本质上是引用传递，但是会产生值传递的效果）。例如：


```

1  # 作用等同于 def funcvar(x): return x + 1
2  funcvar = lambda x: x + 1
3  >>> print funcvar(1)
4  2
5
6  # an_int 和 a_string 是可选参数，它们有默认值
7  # 如果调用 passing_example 时只指定一个参数，那么 an_int 缺省为 2，a_string 缺省为 A default string
8  # a_list 是必备参数，因为它没有指定缺省值。
9  def passing_example(a_list, an_int=2, a_string="A default string"):
10     a_list.append("A new item")
11     an_int = 4
12     return a_list, an_int, a_string
13
14 >>> my_list = [1, 2, 3]
15 >>> my_int = 10
16 >>> print passing_example(my_list, my_int)
17 ([1, 2, 3, 'A new item'], 4, "A default string")
18 >>> my_list
19 [1, 2, 3, 'A new item']
20 >>> my_int
21 10

```

类

Python支持有限的多继承形式。私有变量和方法可以通过添加至少两个前导下划线和最多尾随一个下划线的形式进行声明（如“__spam”，这只是惯例，而不是Python的强制要求）。当然，我们也可以给类的实例取任意名称。例如：

```

1  class MyClass(object):
2      common = 10
3      def __init__(self):
4          self.myvariable = 3
5      def myfunction(self, arg1, arg2):
6          return self.myvariable
7
8      # This is the class instantiation
9      >>> classinstance = MyClass()
10     >>> classinstance.myfunction(1, 2)
11     3
12     # This variable is shared by all classes.
13     >>> classinstance2 = MyClass()
14     >>> classinstance.common
15     10
16     >>> classinstance2.common
17     10
18     # Note how we use the class name
19     # instead of the instance.
20     >>> MyClass.common = 30
21     >>> classinstance.common
22     30
23     >>> classinstance2.common
24     30

```

```

25 # This will not update the variable on the class,
26 # instead it will bind a new object to the old
27 # variable name.
28 >>> classinstance.common = 10
29 >>> classinstance.common
30 10
31 >>> classinstance2.common
32 30
33 >>> MyClass.common = 50
34 # This has not changed, because "common" is
35 # now an instance variable.
36 >>> classinstance.common
37 10
38 >>> classinstance2.common
39 50
40
41 # This class inherits from MyClass. The example
42 # class above inherits from "object", which makes
43 # it what's called a "new-style class".
44 # Multiple inheritance is declared as:
45 # class OtherClass(MyClass1, MyClass2, MyClassN)
46 class OtherClass(MyClass):
47     # The "self" argument is passed automatically
48     # and refers to the class instance, so you can set
49     # instance variables as above, but from inside the class.
50     def __init__(self, arg1):
51         self.myvariable = 3
52         print arg1
53
54 >>> classinstance = OtherClass("hello")
55 hello
56 >>> classinstance.myfunction(1, 2)
57 3
58 # This class doesn't have a .test member, but
59 # we can add one to the instance anyway. Note
60 # that this will only be a member of classinstance.
61 >>> classinstance.test = 10
62 >>> classinstance.test
63 10

```

异常

Python中的异常由 try-except [exceptionname] 块处理，例如：

```

1  def some_function():
2      try:
3          # Division by zero raises an exception
4          10 / 0
5      except ZeroDivisionError:
6          print "Oops, invalid."
7      else:
8          # Exception didn't occur, we're good.
9          pass
10     finally:
11         # This is executed after the code block is run
12         # and all exceptions have been handled, even
13         # if a new exception is raised while handling.
14         print "We're done with that."
15
16 >>> some_function()
17 Oops, invalid.
18 We're done with that.

```

导入

外部库可以使用 `import [libname]` 关键字来导入。同时，你还可以用 `from [libname] import [funcname]` 来导入所需要的函数。例如：

```
1 import random
2 from time import clock
3
4 randomint = random.randint(1, 100)
5 >>> print randomint
6 64
```

文件I / O

Python针对文件的处理有很多内建的函数库可以调用。例如，这里演示了如何序列化文件(使用pickle库将数据结构转换为字符串)：

```
1 import pickle
2 mylist = ["This", "is", 4, 13327]
3 # Open the file C:\\binary.dat for writing. The letter r before the
4 # filename string is used to prevent backslash escaping.
5 myfile = open(r"C:\\binary.dat", "w")
6 pickle.dump(mylist, myfile)
7 myfile.close()
8
9 myfile = open(r"C:\\text.txt", "w")
10 myfile.write("This is a sample string")
11 myfile.close()
12
13 myfile = open(r"C:\\text.txt")
14 >>> print myfile.read()
15 'This is a sample string'
16 myfile.close()
17
18 # Open the file for reading.
19 myfile = open(r"C:\\binary.dat")
20 loadedlist = pickle.load(myfile)
21 myfile.close()
22 >>> print loadedlist
23 ['This', 'is', 4, 13327]
```


其它杂项

- 数值判断可以链接使用，例如 $1 < a < 3$ 能够判断变量 a 是否在1和3之间。
- 可以使用 `del` 删除变量或删除数组中的元素。
- 列表推导式(List Comprehension)提供了一个创建和操作列表的有力工具。列表推导式由一个表达式以及紧跟着这个表达式的for语句构成，for语句还可以跟0个或多个if或for语句，来看下面的例子：

```
1 >>> lst1 = [1, 2, 3]
2 >>> lst2 = [3, 4, 5]
3 >>> print [x * y for x in lst1 for y in lst2]
4 [3, 4, 5, 6, 8, 10, 9, 12, 15]
5 >>> print [x for x in lst1 if 4 > x > 1]
6 [2, 3]
7 # Check if an item has a specific property.
8 # "any" returns true if any item in the list is true.
9 >>> any([i % 3 for i in [3, 3, 4, 4, 3]])
10 True
11 # This is because 4 % 3 = 1, and 1 is true, so any()
12 # returns True.
13
14 # Check how many items have this property.
15 >>> sum(1 for i in [3, 3, 4, 4, 3] if i == 4)
16 2
17 >>> del lst1[0]
18 >>> print lst1
19 [2, 3]
20 >>> del lst1
```

全局变量在函数之外声明，并且可以不需要任何特殊的声明即能读取，但如果你想要修改全局变量的值，就必须在函数开始之处用`global`关键字进行声明，否则Python会将此变量按照新的局部变量处理（请注意，如果不注意很容易被坑）。例如：

```
1 number = 5
2
3 def myfunc():
4     # This will print 5.
5     print number
6
7 def anotherfunc():
8     # This raises an exception because the variable has not
9     # been bound before printing. Python knows that it an
10    # object will be bound to it later and creates a new, local
11    # object instead of accessing the global one.
12    print number
13    number = 3
14
15 def yetanotherfunc():
16     global number
17     # This will correctly change the global.
18     number = 3
```

小结

本教程并未涵盖Python语言的全部内容(甚至连一小部分都称不上)。Python有非常多的库以及很多的功能特点需要学习，所以要想学好 Python你必须在此教程之外通过其它方式，例如阅读Dive into Python。我希望这个教程能给你一个很好的入门指导。

原译文链接：<http://blog.jobbole.com/43922/>

原文链接：<http://www.stavros.io/tutorials/python/#>

关于Python编程的一些问答

作者：赖勇浩

导语

大约1个月前，oschina.net和华章图书一起合作做了一个活动：OSC第51期高手问答——聊聊python那些事，来推广我参与撰写的书《编写高质量代码：改善Python程序的91个建议》（豆瓣链接）。在回答问题的过程中，我看到有若干问题是好几个人都问了的，就萌发了在事后把这些问答整理整理的想法，以下内容就是来自那一次的问答。为简化整理，已经去掉了提问人的昵称，并做了简单的分类。

纠结的Py2与Py3之选

Q:Python 3 会导致Python 的什么前景？最终由3统一，还是一直分裂？各自用自己的版本？

A:py3 自发布以来，进步很大，解决了很多py2无法解决的问题，所以我是坚信大家最终会转到py3。现在py3已经有许多非常吸引人的特性，比如yield from，比如asyncio，比如更漂亮的库结构，等等。但我承认py3还没有一个巨大的吸引让大家转过去，yield from 算半个，另外半个我觉得是jit，想像一下py3自带jit，运行速度是py2的3到10倍，大家肯定一窝蜂转过去了。我觉得py3是未来，但也赞同两个 割裂的版本影响推广。

Q:新手入门应该学2还是3？2会被Python团队放弃吗？

A:学py2吧，如果到时要转py3也是很容易的事。但如果直接学py3，到时候项目要用py2，就会觉得由奢入俭，很痛苦。

Q:请问您觉得Python3.x需要多久才能成为主流?

A:还有比较长的时间,但如果py3.5有独占的、可靠的、官方的jit方案的话,应该会加速很多很多!

怎样学习Python

Q:Python适合作为一个编程入门语言吗。

A:考虑到就业等,我觉得C语言还是更适合作为入门。

Q:python学习的进阶? 基本知识掌握后,该怎么学习?

A:如果已经在工作了,那就直接尝试用python去解决工作需求就好了;如果是学生,那就去复制已存在的网站,它的功能都弄来,比如oschina。

Q:高质量的代码是怎么写的,怎么提升自己的代码的质量?

A:个人看法: 1、熟悉语言的细节; 2、熟悉语言和库的最佳实践; 3、多看一些提升代码质量的指导书籍; 4、同行评审; 5、多学几门语言,博采众长。

Q:对一个java开发者来说学习使用python有什么好的建议吗?

A:像当年学习java一样学习它。

Q:初学者,有什么好的网站推荐学习?

A:还是读书、看手册吧,网站的知识太零散,不成体系,容易学成野路子程序员。

Q:学习和使用了一段python后怎么做才能更好的提高自己使用python的能力呢?

A:我的回答是复刻一些产品，比如自己尝试做个豆瓣、oschina之类。

Q:《编写高质量代码：改善Python程序的91个建议》这本书适合其它非Python程序员看吗?

A:不适合。它的定位是对python有所了解的人。给初中级python程序员提升到中高级，这样的定位。

Q:请推荐几个比较优秀的Python开源项目，用来学习的。

A:优秀的python开源项目啊，我想一下，额，看一下trac和reviewboard?

Q:你觉得你学Python以来觉得关于Python最好的书籍有哪些呢？语言基础、语言设计、框架方面，等等都行。谢谢了。

A:很好的问题，我觉得可以回答，因为我自己觉得自己的书也不是“最好的”那个层次。推荐《expert python programming（中译Python高级编程，翻译差强人意）》、《Python源码剖析》。

Q: 你好，我有个问题想请教你，如何成为真正的pythoner。我以前是做C++的，现在工作会接触一些python web方面的任务。我最开始接触python的时候学了些基本的语法就开始尝试写与算法相关的程序了，所以写python程序的时候代码还是有C++的影子，直到现在工作中维护之前别人的python web程序的时候，发现自己的python程序太不优雅了，好多时候有的功能可以用更美更快更高效的方式实现，但我可能会像C++里面一样想要从底层一点一草一木的搭建、控制我的程序，后来发现可以几行甚至更短的代码就可以实现那些功能，而且执行效率也更好一点，于是发现写python程序不能用以前C++代码时的思想，或许我可以尝试更pythoner一点？

A:很有意思的一个问题，个人建议如下：1、通读一遍手册，特别是lib ref和lang ref，所谓熟能生巧，对语言本身的熟悉才能写得更pythonic；2、多看一下经典的python项目的文档，比如flask，比如pip，看看他们提供了什么机制，甚至探究到这样的机制是怎么提供的；3、不要担心，多写一些python代码就好了。

应用Python的困惑

Q:Python 适合开发比较复杂的web项目吗？

A:在我看来，python适合开发小中大巨多种复杂程度的项目，因为能不能把项目做成，最关键的因素还是人。python对web支持不错，有许多第三方库，也有django/flask等许多人叫好的框架。

Q:入门Python用哪个操作系统环境好？我就只在Windows上学过一点基础，但不知一般真实团队开发环境是怎样，可以可以介绍下？谢谢！

A:我们一直用linux作为生产环境，mac os x 是我的开发环境，我用过许多年的windows，我不觉得使用windows会影响你的学习。

Q:目前python似乎大多数都是在WEB方向的应用，对于做C语言和C++的开发python有什么可以帮助的地方

A:主要是利用动态语言的灵活性、解释型语言的方便性，来解决C/C++在应变多变的业务需求、快速部署等方面的成本过高或力不从心的问题。

经验之谈

Q:学生，只往PyQt方向发展可行吗？

A:从长远来说，一里通百里融，只要有个突破口，后面有成就也是很可能的。从短期来说，感觉这方面就业面比较窄。

Q:感觉python什么都可以做，但是很难做到很好。如果没有c/c++/go扩展，python能做出大吞吐量，高并发高稳定性的系统吗

A: 说python什么都可以做，还是高估了，但说python很难做到很好，就得看“很好”这个词怎么定义了，估计各人的标准还不一样。恕我见识少，好像用go来扩展python还没有成熟方案？目前我经历的网游、网站方面来说，我觉得还可以做出大吞吐量的高并发系统的，只是可能硬件成本会稍高些，至于稳定性，决定性的因素是开发人员的质量，跟语言关系不大，甚至c/c++更容易开发出不稳定的系统。

Q:python 的类库，函数库庞大，如何能快速找到自己需要的类库？

A:这是一个问题，很多语言都有同样的问题。要不你先去看一下awesome-python 这个项目？

Q:python程序员收入咋样

A:好像年收入从几万到几十万的都见过，但上百万的我还没有见过。总的来说，我觉得跟其它语言的差不多，但高薪的比例可能是比较靠前的。

Q: 我原是一名ruby程序员，后来看到python有非常多的模块，如ipython、ipython notebook等非常酷的python应用，其实在灵活性上python远不如ruby（method_missing等），设计哲学也不一样，想问一下为什么python比ruby的应用多很多？特别是科学计算（Numpy）和绘图（matplotlib）？

A:于python会在科学界这么流行的原因，据说是因为那些科学家都不是计算机专业的，觉得python这货容易学容易用，所以就用这个，反正只是一次性地写写脚本跑一下试验数据。

Q: 我想请问，python用来写游戏好么，大家不都说python相对运行起来比c++/c慢，而现在大部分游戏引擎也都是c/c++，就看那个Cocos2d，最先貌似是python版，之后还是用c/c++重写，虽说现在Python版也有更新；另

外，现在移动端的各种软件游戏开发也比较热门，python在这方面好像有点跟不上的感觉。

A:是，在手游时代，python已经不合适编写客户端了。可以尝试在服务器端使用它，还是很好的。

Q: 我有这么几个问题想请教一下：1. 多框架、多模块的实现语言代表着多学习成本吗2. 胶水语言的存在，现在主要用来做什么，其他层面上是如何应用的3. 类 Unix 系统管理，如何学习他们4. Web 方面与 ruby 之类的有哪些区别，或者说，优缺点5. 写过文字性的爬虫，感觉字符集和平台差异稍有异样，请问您是如何避免的

A:1. 我不太明白多框架、多模块是什么意思，我可以理解为python有很多框架吗？如果是的话，我觉得并不代表更多的学习成本，你看一下quickstart，看一下examples，看一下doc里关于扩展和分拆的机制，就知道这个框架、模块是否适合你的技术观，适合的就行，不适合的就不学，不学不会有成本；2. 胶水语言的这个问题，我觉得《unix编程艺术》这本书里谈得比我讲要好得多，推荐看；3. 关于sa这一块，我了解不多，建议查阅专著；4. ruby的好处就是有ror这样的大一统解决方案，python是选择多，麻烦也多；4. 文本处理，在编码上的问题很多、很难，建议使用 chardet 等方案，但都是有力不能及的地方的，建议接受部分问题是无法解决的事实。

Q: 以下的几个问题想请教一下的：(1) 如何做到高效的python coder, 我也是近这2年才开始用python的，但觉得开化效率还有很多改善的地方，与相对公司内部的java组来说。(2) python 的其中一点我比较喜欢的是比较明了，什么东西都比较原生态，相当于.net来说过于包装，这就是它的长处，如果现成的lib也可以找到，请问一下平时一般常用的东西，是自己做成lib好，还是直接用别人写好的lib好？谢谢(3) python是否比较适合写web呢或是比较好的选择？相对于国内这个环境来说，python这一方面真的比较少些，当然国外有google做带头大哥，它期下的很多projects都很出彩的，我现在做一个web 方面的project，选择bottle，它和flask

比较类似，速度方面会较好一点，发现项目进行中遇到不少关于技术方面的问题，可查找到的资料比较少，解决问题相当困难。

A:1. 怎么做高效的python coder，我觉得跟其它语言没有二致，不管怎么样，多读官方文档肯定大有增益，这也是我的经验之谈。2. 我一般直接用别人写好的lib，如果有而且用起来爽的话；3. 建议选择flask/django等社区比较大、比较活跃的框架。

编码一线

Q:个人感觉python没有成熟的IDE，写程序都是文档不停不停地翻，不想写C++或用eclipse，都是自动提示+文档，我想知道是我实在是才疏学浅还是python开发的通病呢？

A:嗯，其实没有特别厉害的IDE，pyCharm是比较好的，而且有免费的社区版，习惯了用E记的，可以试一下pydev。

Q:想问一下，python每import一个模块都会在内存中实例化还是共享一个实例。

A:默认是共享一个。

Q:您是用什么工具打包python工程的？我使用pyinstaller打包包含gtk的程序后，执行打包后的文件会报_glib module 不存在。不知道还有什么更好的工具没。最好是跨平台的。

A:我们一般不打安装包，我们一般就是用setuptools。

Q:python“函数”的返回值类型不在语句的语法声明中，使用python开发项目，由于返回值类型不那么明确，怎么确保在软件迭代中不会弄错？

A:一方面是大家遵守一些最佳实践，比如保证返回值都是同一类型的；二是充分进行单元测试；三是使用较新的python版本，已经支持参数和返回值的类型声明。

Q:对于python 的协程 有什么好的库比较好用

A:必须是gevent。

Q:请问Python有类似Ruby社区RVM、Bundler、Rake的成熟工具链么？

A:有的，pypi.python.org，你可以上去看一下。我书里也有提到这些工具链的用法。

原文链接：<http://blog.csdn.net/gzlayonghao/article/details/38985017>

JavaScript构建（编译）系统大比拼

作者：Nicolas Bevacqua

Nicolas Bevacqua进行了一个比较JavaScript 构建（编译）系统的任务。他对三巨头： Grunt, Gulp and NPM进行了比较，并讨论了每种优缺点。

决定采用何种技术总是很难的。一旦遇到问题，你不想推翻你之前的选择。但是你必须选一个，然后让它按照着你的思路做。实施一套构建（编译）系也是一样的，你应该把它看作一个非常重要的选择，让我们以Grunt为例。

- Grunt有一个完善的社区，即使是在Windows上
- 它不仅仅应用在Node社区
- 它简单易学，你可以随便安装插件并配置它们
- 你不需要多先进的理念，也不需要任何经验

这些都是用Grunt构建编译工具的充分理由，但我想澄清一点，我不认为Grunt不是唯一最好的选择。还有一些同样流行的选择摆在那里，有些方面可能比Grunt做得更好。

我写这篇文章，以帮助您了解Grunt，Gulp和npm之间的差异，这是我在前端开发工作中使用最多的三种构建工具。

我们先来讨论Grunt擅长的方面

Grunt: 好的部分

Grunt 最好的一个方面是它的易用性。它能使程序员使用JavaScript构建编译工具时，几乎不费吹灰之力。你只需要寻找合适的插件，阅读它们的文档，然后安装和配置它。这种易用性意味着大型开发团队，那些不同技能水平的成员，也可以没有任何麻烦的调整编译流程，以满足项目的最新需求。而且团队并不需要精通 Node，他们仅需要配置对象，将不同的任务添加到不同的序列构建编译流程。

这里有基础足够大的插件库，你会发现自己几乎不需要开发自己的编译任务，这能使您和您的团队能够快速构建开发工具，如果你要快速完成编译过程这是至关重要的，你也可以采取小步走，逐步完善编译流程的策略。

通过Grunt管理部署也是可行的，因为有许多包已经可以完成这些任务，如 `grunt-git`, `grunt-rsync`, 或 `grunt-ec2` 等等。

那么，Grunt有什么缺陷吗？如果你有一个明显复杂的编译过程，它可能会变得过于冗长。当开发一段时间以后，它往往很难将编译过程作为一个整体。一旦你编译流程任务到达两位数，几乎可以保证，你会发现自己不得不在多个目标(Targets)中跑同一个Task，以便你能够正确地执行任务流。由于任务是需要声明配置的，你也很难弄清楚任务真正的执行次序

除此之外，你的团队应该致力于编写可维护的代码，当涉及到你的编译，比如在使用Grunt的情况下，这意味着你需要为每个任务（或者每个编译流）编写一份独立的配置文件，供你的团队使用。

现在，我们已经了解了Grunt好和不好的方面，以及在何种情况下，比较适合作为你项目的编译工具。我们再来谈谈npm，它如何被用作构建工具，以及与Grunt有何不同。

将npm视为构建工具

为了将NPM用作构建工具，你需要一个package.json和 npm。制定NPM任务就像在脚本中添加属性一样简单。该属性的名称将用作任务名和将要执行的命令。下面的这个build任务将预先检查我们的 JavaScript代码中有没有语法错误，例子使用JSHint命令行接口来。在命令行中你可以运行任何你需要的shell。

```
{
  "scripts": {
    "test": "jshint . --exclude node_modules"
  },
  "devDependencies": {
    "jshint": "^2.5.1"
  }
}
```

一旦定义完成，就可以通过下面的命令来运行

```
npm run test
```

需要注意的是npm提供了运行特定任务的快捷方式。比如要运行test，你可以简单地使用npm test并省略动词run。您可以通过一个命令链来将一系列npm run的任务连在一起，构成你的编译流程：

```
{
  "scripts": {
```

```

    "lint": "jshint . --exclude node_modules",
    "unit": "tape test/*",
    "test": "npm run lint && npm run unit"
  },
  "devDependencies": {
    "jshint": "^2.5.1",
    "tape": "~2.10.2"
  }
}

```

您 也可以安排一些后台完成的任务，然后让他们同步。假设我们有以下的包文件，我们将复制出一个目录用来放JavaScript文件，以及将我们用Stylus写的样式表文件编译成CSS。在这种情况下，多个任务一起运行是比较理想的。也可以实现，使用&分隔符即可。

```

{
  "scripts": {
    "build-js": "cp -r src/js/vendor bin/js",
    "build-css": "stylus src/css/all.styl -o bin/css",
    "build": "npm run build-js & npm run build-css"
  },
  "devDependencies": {
    "stylus": "^0.45.0"
  }
}

```

要了解关于将npm用作构建工具的更多内容，你应该先学学写一些Bash命令。

安装NPM的任务依赖

JSHint CLI并不一定要包含在你的系统中，这里有两种安装它的方式。如果你正在寻找直接从命令行中运行的工具，那么你应该在全局范围内安装，使用g标志，如下所示。

```
npm install -g jshint
```

不过，如果您使用的是包在npm run中使用的，那么你就应该把它加到devDependency中，如下所示。这将让npm自动在系统中寻找它所依赖的JSHint安装在了哪里。这方法适用于任何命令行工具中和所有操作系统。

```
npm install --save-dev jshint
```

你其实不仅局限使用CLI工具。事实上，npm能够运行任何shell脚本。让我们来挖一挖！

在npm中使用shell脚本

下面是一个运行在node的脚本，并显示一个随机的绘文字符串(emoji-random)。第一行指定运行环境，该脚本基于Node。

```
#!/usr/bin/env node  
  
var emoji = require('emoji-random');  
  
var emo = emoji.random();  
  
console.log(emo);
```


如果你将一个名为emoji的脚本文件放到你项目的根目录中，你必须将emoji-random申报为依赖关系，并将以下脚本命令添加到包文件中。

```
{
  "scripts": {
    "emoji": "./emoji"
  },
  "devDependencies": {
    "emoji-random": "^0.1.2"
  }
}
```

一旦写成这样，你只需要在命令行运行 `npm run emoji` 即可。

好和坏的方面

使用NPM作为构建工具比Grunt有几大优势。你不会被Grunt的插件束缚，你可以利用NPM的所有优势，它有数以万计的模块可以选择。除了NPM，你不需要任何额外的命令行工具(CLI)或文件，你只需要在package.json添加依赖关系。由于NPM运行命令行工具(CLI 工具)和Bash命令，这比Grunt执行的方式更好。

Grunt 的最大缺点之一就是它的I/O限制。这意味着大多数Grunt的任务将从磁盘中读取，再写入到磁盘。如果你的多个任务需要操作同一个文件，那么该文件很有可能被从磁盘中多次读取。在bash中，命令通过管道直接传递给下一个任务，避免Grunt额外的I/O开销。

也许NPM的最大的缺点是，在Windows环境中的应用可能没那么好。这意味着使用NPM运行的开源项目可能遇到问题。这也意味着Windows开发人员尝试使用npm的替代品。这缺点几乎将NPM从Windows上排除。

Gulp，另一个构建工具，提出了与Grunt和npm相似的功能，一会你就会发现。

Gulp的流式构建工具

与 Grunt类似，它依赖插件，并且是跨平台的，它也支持Windows。Gulp是一个代码驱动的构建工具，与Grunt的声明式定义任务相反，它的任务定义更容易阅读一点。Gulp也有类似于npm run的东西，因为它使用Node Stream来转化输入输出。这意味着，Gulp没有Grunt那种磁盘密集型I/O操作的问题。它也是它比Grunt更快的原因，更少的时间花在I/O 上面。

在使用Gulp的主要缺点是，它在很大程度上依赖于流，管道和异步代码。不要误解我的意思：如果你用在Node中，这绝对是一个优势。但是，除非你和你的团队非常精通Node，你很有可能会遇到处理流的问题，特别是如果你要建立你自己的Gulp任务插件。

在团队工作的时候，Gulp不是望而却步的npm，因为大多数前端团队可能都懂JavaScript，但是他们可能对Bash脚本不那么熟练，其中一些可能是使用Windows的！这就是为什么我通常建议你在个人项目中运行NPM的原因。如果你的团队很熟悉Node，你可以使用Gulp。当然，这是我个人的建议，你应该找到最适合你和你团队的工具。此外，你应该不会把自己限制在Grunt，Gulp，或者npm run中，对你我来说这些都只是工具。尝试做一些小小的研究，也许你会发现，你喜欢的甚至比这三个更好的工具。

让我们通过一些例子来看看Gulp中的任务看起来是什么样子的。

在Gulp中运行测试

有一些约定Gulp与Grunt极为相似。在Grunt中有一个定义Task的文件Gruntfile.js，在Gulp中叫Gulpfile.js。另一种微小的差别是，在Gulp中，CLI已经包含在同一个Gulp包中，你需要通过npm从本地和全局同时安装。

```
touch Gulpfile.js
```

```
npm install -g gulp
```

```
npm install --save-dev gulp
```

在开始之前，我将创建一个Gulp任务处理一个JavaScript文件，就像你已经在Grunt和NPM中看到的那样使用JSHint，你需要先安装gulp-jshint，Gulp的JSHint插件。

```
npm install --save-dev gulp-jshint
```

现在你已经同时在全局和本地中装好CLI了，本地已经安装了gulp和gulp-jshint插件，你可以将这些构建任务合成一个。你可以在Gulpfile.js文件中写出来。

首先，您将使用gulp.task定义一个任务和功能。该功能包含了所有必要的代码来运行这项测试。在这里，你应该使用gulp.src创建一个读取你源文件的流，这个数据流会被管道输送进JSHint插件。然后，所有你需要做的就是管道中的JSHint任务打印到终端。下面是Gulpfile中展示的结果。

```
var gulp = require('gulp');  
var jshint = require('gulp-jshint');  
gulp.task('test', function () {  
  return gulp  
    .src('./sample.js')  
    .pipe(jshint())  
    .pipe(jshint.reporter('default'));  
});
```


还有一点需要提一下，Gulp流会在一个任务完全结束之后再转到下一个任务。你可以使用一个JSHint Reporter使输出更加简洁，从而更易于阅读。JSHint Reporter并不需要Gulp插件，例如jshint-stylish，让我们在本地直接安装。

```
npm install --save-dev jshint-stylish
```

更新后的Gulpfile应如下所示。它会加载jshint-stylish模块，按报表格式输出。

```
var gulp = require('gulp');  
var jshint = require('gulp-jshint');  
gulp.task('test', function () {  
  return gulp  
    .src('./sample.js')  
    .pipe(jshint())  
    .pipe(jshint.reporter('jshint-stylish'));  
});
```

大功告成！这是所有一个命名为test的Gulp的任务。它可以使用下面的命令运行，只要你安装了全局的CLI。

```
gulp test
```

这是一个相当简单的例子。你也可以通过使用gulp.dest，创建了一个写数据流到磁盘中。让我们看看另外一个构建任务。

在Gulp中创建一个库

在开始之前，让我们明确任务：从磁盘gulp.src读取源文件并通过磁盘管道写回内容到gulp.dest，你可以理解成只是将文件复制到另一个目录。

```
var gulp = require('gulp');  
gulp.task('build', function () {  
  return gulp  
    .src('./sample.js')  
    .pipe(gulp.dest('./build'));  
});
```

复制文件完成了，但是它没有压缩这个JS文件。要做到这一点，你必须使用一个Gulp插件。在这种情况下，你可以使用gulp-uglify，流行的UglifyJS压缩编译插件。

```
var gulp = require('gulp');  
var uglify = require('gulp-uglify');  
gulp.task('build', function () {  
  return gulp  
    .src('./sample.js')  
    .pipe(uglify())  
    .pipe(gulp.dest('./build'));  
});
```

正如你可能意识到的那样，流使可以让您添加更多的插件，而只需要读取和写入磁盘一次。你也可以指定缓冲器中内容的大小。需要注意的是，如果你在压缩之前添加它，那么你得到的大小是unminified。

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');
var size = require('gulp-size');
gulp.task('build', function () {
  return gulp
    .src('./sample.js')
    .pipe(uglify())
    .pipe(size())
    .pipe(gulp.dest('./build'));
});
```

为了增强这种组合，满足添加或删除管道的需要，让我们添加最后一个插件。这一次，我会用gulp-header在头文件添加一段版权信息的代码，如名称，版本和许可证类型。

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');
var size = require('gulp-size');
var header = require('gulp-header');
var pkg = require('./package.json');
```



```
var info = '// <%= pkg.name %>@v<%= pkg.version %>, <%=  
pkg.license %>\n';
```

```
gulp.task('build', function () {  
  return gulp  
    .src('./sample.js')  
    .pipe(uglify())  
    .pipe(header(info, { pkg : pkg }))  
    .pipe(size())  
    .pipe(gulp.dest('./build'));  
});
```

就像Grunt一样，在Gulp中你可以通过传递一组任务到gulp.task来定义流程。在这方面，Grunt和Gulp之间的主要区别在于，Grunt是同步的，而Gulp是异步的。

```
gulp.task('build', ['build-js', 'build-css']);
```

在Gulp，如果你要让任务同步运行，你必须声明一个任务。你的任务开始之前执行。

```
gulp.task('build', ['dep'], function () {  
  // 执行dep所依赖的任务  
});
```

如果你有任何收获，先看看这段话。

你使用哪种工具并不重要，只要保证：流程构建（编译）好用就行了，用起来不要太辛苦。

原译文链接： <http://ourjs.com/detail/53f2be04c1afbc6e30000001>

原文链接： http://modernweb.com/2014/08/04/choose-grunt-gulp-npm/?utm_source=ourjs.com

为什么Android的图片质量会比iPhone的差？

作者：比太钱包

经常看到有人问：“安卓版微信发出去的图片怎么那么渣！比iPhone的差远了！”。不只是微信，很多应用安卓版的图片质量就是要比iPhone版逊色很多，这到底是怎么回事？

我们团队最初也纠结过这个问题，费了半天劲、绕了好大圈，直到最后才发现，原来这是谷歌犯得一个“小”错误，而且一直错到了今天。

谷歌的错就在于：libjpeg。

libjpeg是广泛使用的开源JPEG图像库（参考<http://en.wikipedia.org/wiki/Libjpeg>），安卓也依赖libjpeg来压缩图片。通过查看源码，我们会发现安卓并不是直接封装的libjpeg，而是基于了另一个叫Skia的开源项目（http://en.wikipedia.org/wiki/Skia_Graphics_Engine）来作为的图像处理引擎。Skia是谷歌自己维护着的一个大而全的引擎，各种图像处理功能均在其中予以实现，并且广泛的应用于谷歌自己和其它公司的产品中（如：Chrome、Firefox、Android等）。Skia对libjpeg进行了良好的封装，基于这个引擎可以很方便为操作系统、浏览器等开发图像处理功能。

libjpeg在压缩图像时，有一个参数叫optimize_coding，关于这个参数，libjpeg.doc有如下解释：

```
boolean optimize_coding
```


TRUE causes the compressor to compute optimal Huffman coding tables

for the image. This requires an extra pass over the data and therefore costs a good deal of space and time. The default is FALSE, which tells the compressor to use the supplied or default Huffman tables. In most cases optimal tables save only a few percent

of file size compared to the default tables. Note that when this is TRUE, you need not supply Huffman tables at all, and any you do supply will be overwritten.

这段话大概的意思就是如果设置optimize_coding为TRUE，将会使得压缩图像过程中基于图像数据计算哈弗曼表（关于图片压缩中的哈弗曼表，请自行查阅相关资料），由于这个计算会显著消耗空间和时间，默认值被设置为FALSE。

这段解释乍看起来没有任何问题，libjpeg的代码也经受了十多年的考验，健壮而高效。但很多人忽略了这一点，那就是，这段解释是十多年前写的，对于当时的计算设备来说，空间和时间的消耗可能是显著的，但到今天，这似乎不应再是问题，相反，我们应该更多的考虑图片的品质（越来越好的显示技术）和图片的大小（越来越依赖于云服务）。

谷歌的Skia项目工程师们最终没有设置这个参数，optimize_coding在Skia中默认的等于了FALSE，这就意味着更差的图片质量和更大的图片文件，而压缩图片过程中所耗费的时间和空间其实反而是可以忽略不计的。那么，这个参数的影响究竟会有多大呢？

经我们实测，使用相同的原始图片，分别设置optimize_coding=TRUE和FALSE进行压缩，想达到接近的图片质量（用Photoshop 放大到像素级逐块对比），FALSE时的图片大小大约是TRUE时的5-10倍。换句话说，如果我们想在FALSE和TRUE时压缩成相同大小的JPEG 图片，FALSE的品质

将大大逊色于TRUE的（虽然品质很难量化，但我们不妨说成是差5-10倍）。

我们又对Android和iOS进行了对比（均使用标准的JPEG压缩方法），两个系统都没有提供设置optimize_coding的接口（通过阅读源码，我们已经知道Android是FALSE，iOS不详），当压缩相同的原始图片时，结果也是一样，iOS完胜。想要品质接近，文件大小就会差出5-10倍，而如果要压缩出相同大小的文件，Android的压缩品质简直就是惨不忍睹。

结果说明，苹果很清楚optimize_coding参数和哈弗曼表的意义，这里需要特别指出，苹果使用的哈弗曼表算法与libjpeg（及我们后来自行采用的libjpeg-turbo）不同，像素级可以看出区别，苹果似乎基于libjpeg又进行了进一步的优化，压缩出来的图片细节上更柔和、更平滑。

以上试验，我们尝试过多个原图、多种压缩比例，试验结果均类似，如有兴趣，您不妨也自行进行尝试。

最终我们决定，不再使用安卓系统原生的JPEG压缩方法，而是基于libjpeg-turbo自行编译了一版native的安卓库，专门用来压缩图片，这样在我们的产品中，就做到了仅仅用1/5的图片大小，就能让用户得到不逊色甚至更优的图片品质，对于我们团队来说，费了半天劲、绕了好大圈是非常值得的。（使用libjpeg-turbo还有性能上的好处，这里就不再赘述了）

最后，附上我们团队在github上的开源项目地址，供参考：

<https://github.com/bither>

原文链接：http://blog.sina.com.cn/s/blog_12ce70a430102v1p3.html

一次app抓包引发的Android分析记录

作者：绿盟科技

0×00 起因

最近想了解下移动端app是如何与服务端进行交互的，就顺手下了一个某app抓下http包。谁知抓下来的http居然都长这个模样：

```
POST /ca?qrt=***LoginHTTP/1.1
Content-Type:application/x-www-form-urlencoded
Content-Length:821
Host:client.XXX.com
Connection:Keep-Alive

c=B946D7CF7B9E5B589F8DE6BC9E5DACF08DA6B35DA7A899DCA5AD5A58ADDAD5E66363589
EF2D385B1ACACABDAD5E65F63589EF2D3F5B43EA7ACE36DFCA5383EABE7D4F1403E379FF0
DEFCAD9FABA7E6E1F243A7AAAC74E380A4A09E6593D3FEA4ABA8ACF0DDF0AEAAB3A7EAE9F
BA4A09E5F97D3FEA49EA09E9B97A9A4B69EAD7E1F6B0AC9EA0DA94A95E57609EF2D3B55E
659EA0DA94B55F9EB69EDAD5E668689EB6DA98AA6E576E62939DE6A69E616D868CB673636
162DAEBE6AEA2ACA2E486F3AD9EA09EA898A0A4B69EABE8E1F3B29EA09EA998A0A4B69EAD
E3E385AE3DA7ABDB6FF4B93A9F3DEF3FBA537ACAD77D486AD37ADABE77383B4B4ACB4DAD
5E66E9EB69EA886AF634061599F97E6A69E676394D3FEA4ACACACE8E1F4B2ACACACE8E1F4
B2AC9EA0DA9CAAA4B69E9EDCD3B269589EB6DADFF4B2ACABACE3E9E675&
b=AD178E267CA8666F636D6C54ADC1B3AEAD736574696950776D71A9BEACADA7A8727762A
8C0AA67656172736A6D676F706E6369837F726C71A5AAA4756C656963ADC1A6797870615E
676E7063666C756CAC7E&ext=&v=alex
```


这是我在尝试登陆时抓包获取的唯一http包，显而易见POST数据中的c参数是包含登录信息的，但是为什么长这个模样？为了得到答案，我开启了我一周的Android动态调试和静态分析学习之旅。

这篇文章将通过这段字符串原文的过程，向各位介绍几种非常好用的Android调试工具以及它们的一些简单用法。

0×01 分析过程

1.基本静态分析过程

拿到一个apk最常规的做法应该是就是，反编译查看一下java源码了。

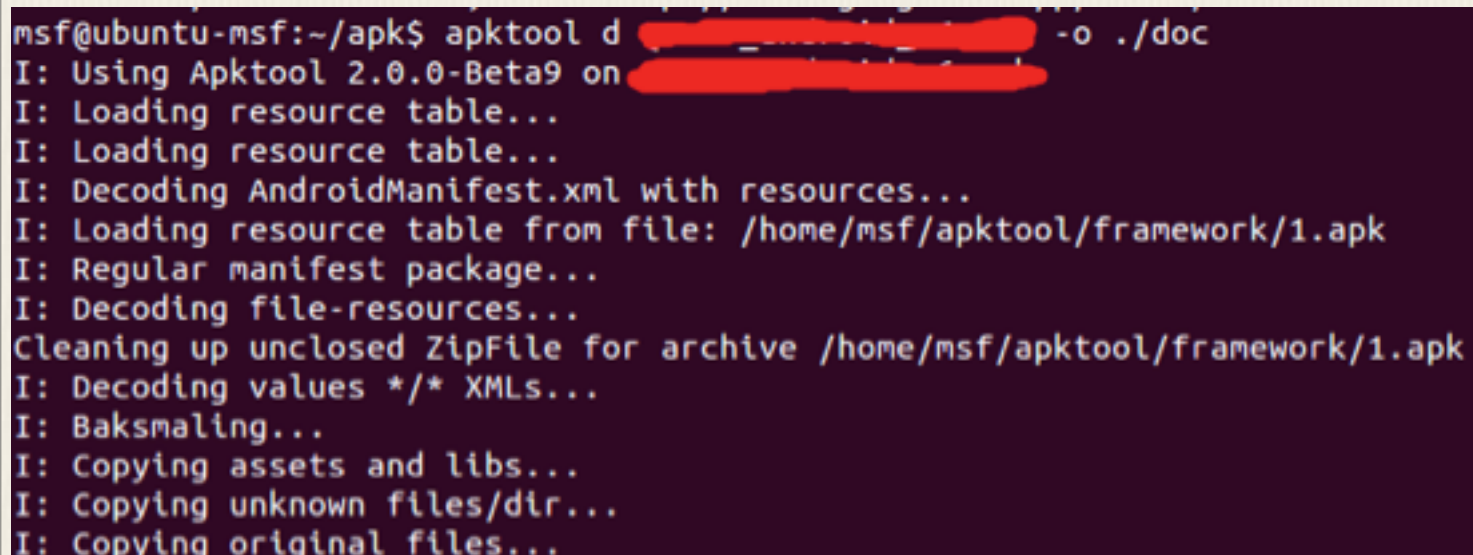
用apktool反编译得到smali（其实主要是为了看AndroidManifest.xml）：

/*我用的apktool是2.0.0-Beta9，命令和常见的1.x版本的命令有所不同*/

```
apktool d XXX.apk -o ./doc
```

/*我用的apktool是2.0.0-Beta9，命令和常见的1.x版本的命令有所不同*/

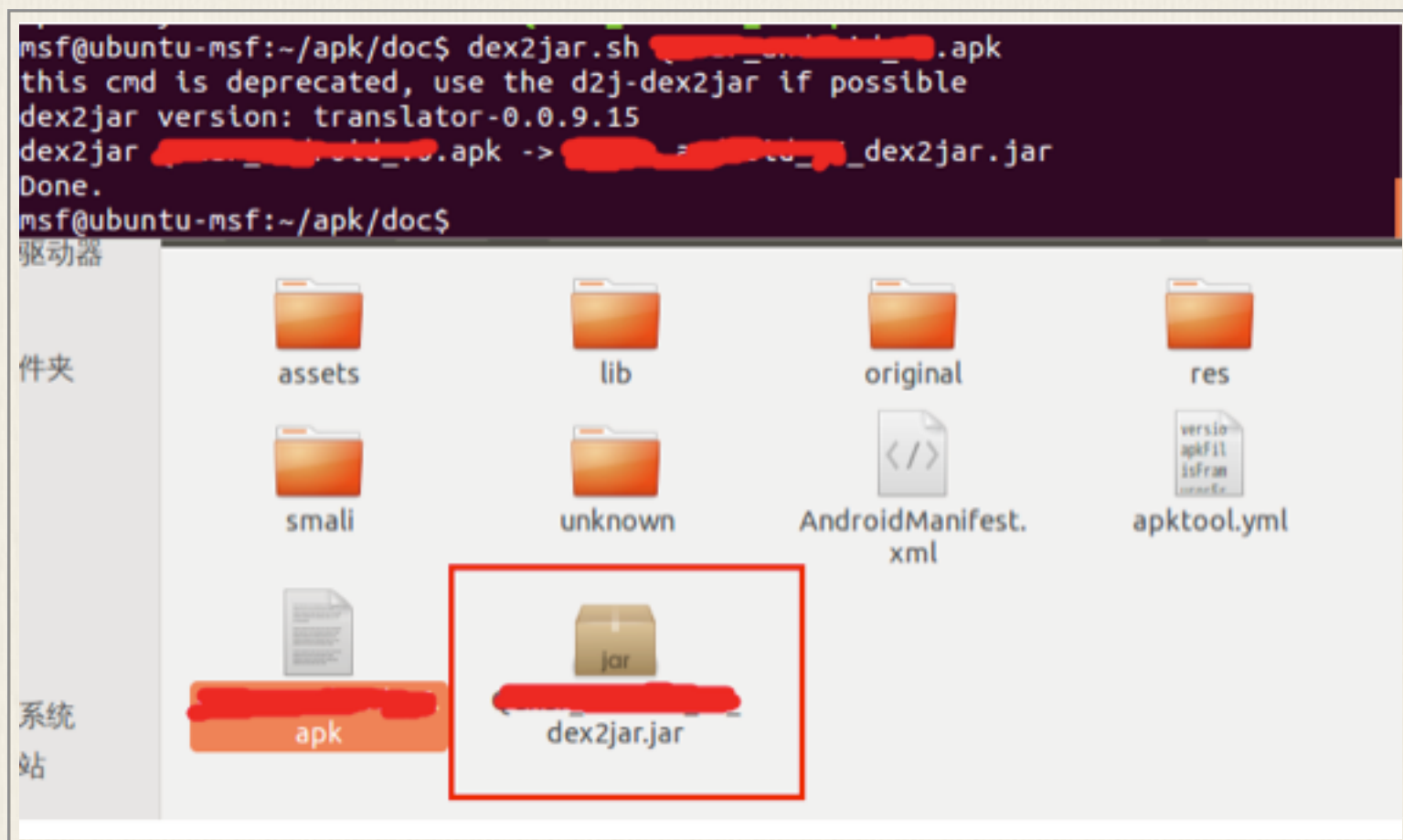
```
apktool d XXX.apk -o ./doc
```

A terminal window screenshot showing the execution of the 'apktool d' command. The command is 'apktool d [redacted] -o ./doc'. The output shows various steps: 'Using Apktool 2.0.0-Beta9 on [redacted]', 'Loading resource table...', 'Decoding AndroidManifest.xml with resources...', 'Loading resource table from file: /home/msf/apktool/framework/1.apk', 'Regular manifest package...', 'Decoding file-resources...', 'Cleaning up unclosed ZipFile for archive /home/msf/apktool/framework/1.apk', 'Decoding values */* XMLs...', 'Baksmaling...', 'Copying assets and libs...', 'Copying unknown files/dir...', and 'Copying original files...'.

```
msf@ubuntu-msf:~/apk$ apktool d [redacted] -o ./doc
I: Using Apktool 2.0.0-Beta9 on [redacted]
I: Loading resource table...
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/msf/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
Cleaning up unclosed ZipFile for archive /home/msf/apktool/framework/1.apk
I: Decoding values */* XMLs...
I: Baksmaling...
I: Copying assets and libs...
I: Copying unknown files/dir...
I: Copying original files...
```

然后用的dex2jar工具将apk反编译为jar，并通过JD-GUI 来查看java源码：

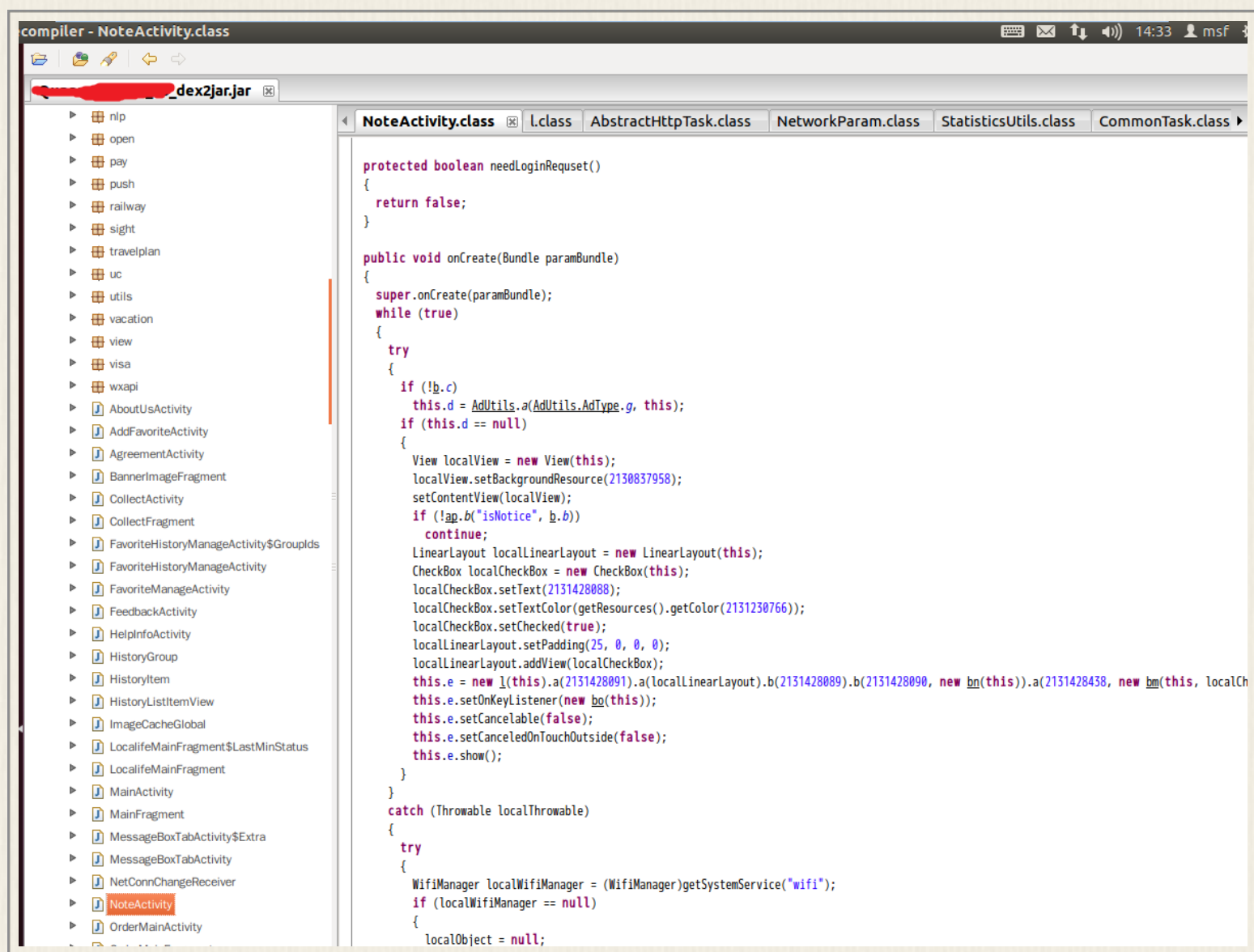
```
dex2jar.sh XXX.apk
```



在apktool反编译的目录中我们可以翻看AndroidManifest.xml来了解apk文件的基本结构，我从中首先找到主Activity的所在：

```
<activityandroid:configChanges="keyboardHidden|orientation"android:export
ed="true"android:name="com.XXX.NoteActivity"android:screenOrientation="po
rtrait"android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen">
<intent-filter><actionandroid:name="android.intent.action.MAIN"
/><categoryandroid:name="android.intent.category.LAUNCHER"
/><categoryandroid:name="android.intent.category.MULTIWINDOW_LAUNCHER"
/></intent-filter>
```

可以从上面的内容看出主Activity是com.XXX.NoteActivity这个类所定义的，然后通过JD-GUI打开dex2jar反编译后的jar包，查看NoteActivity。



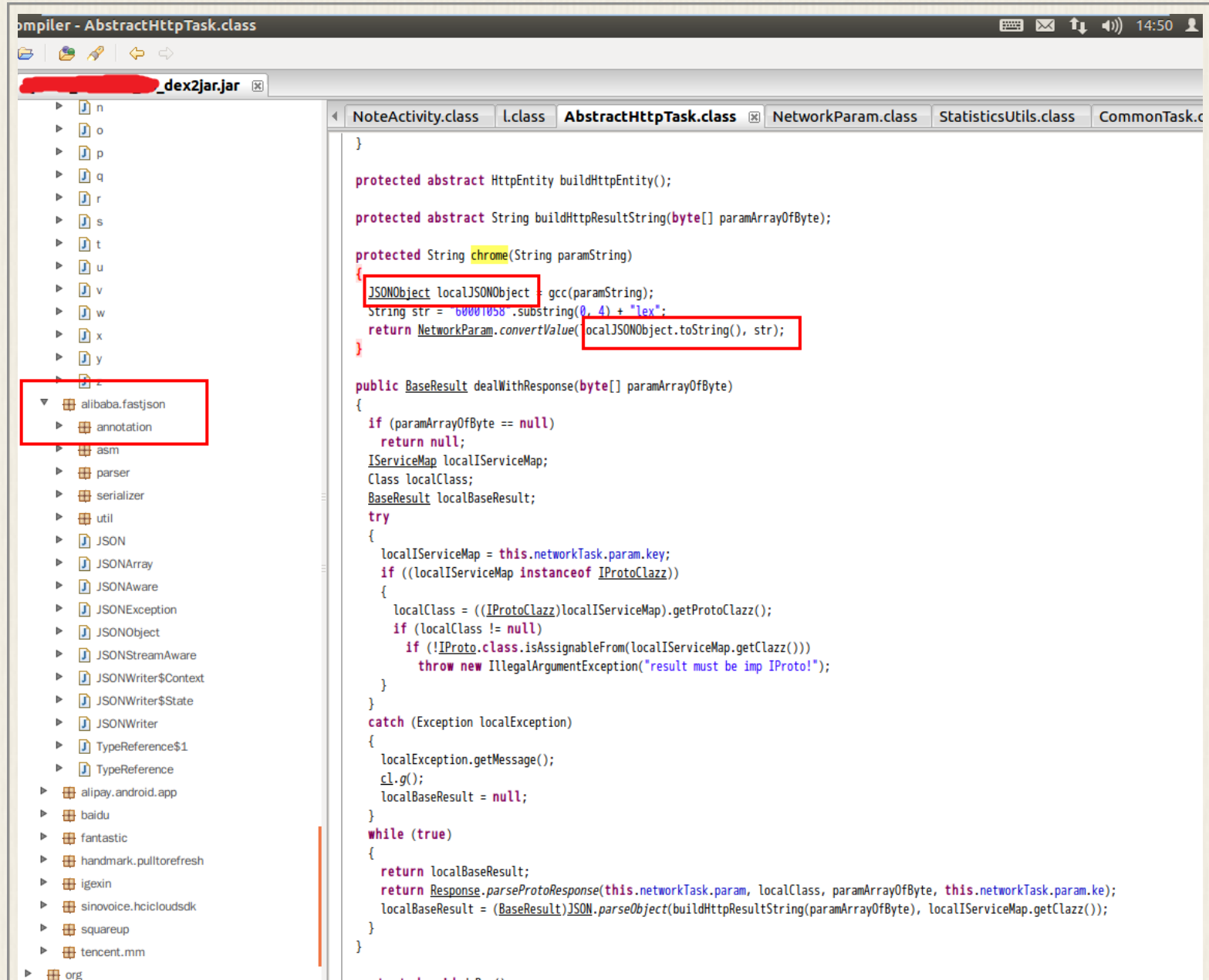
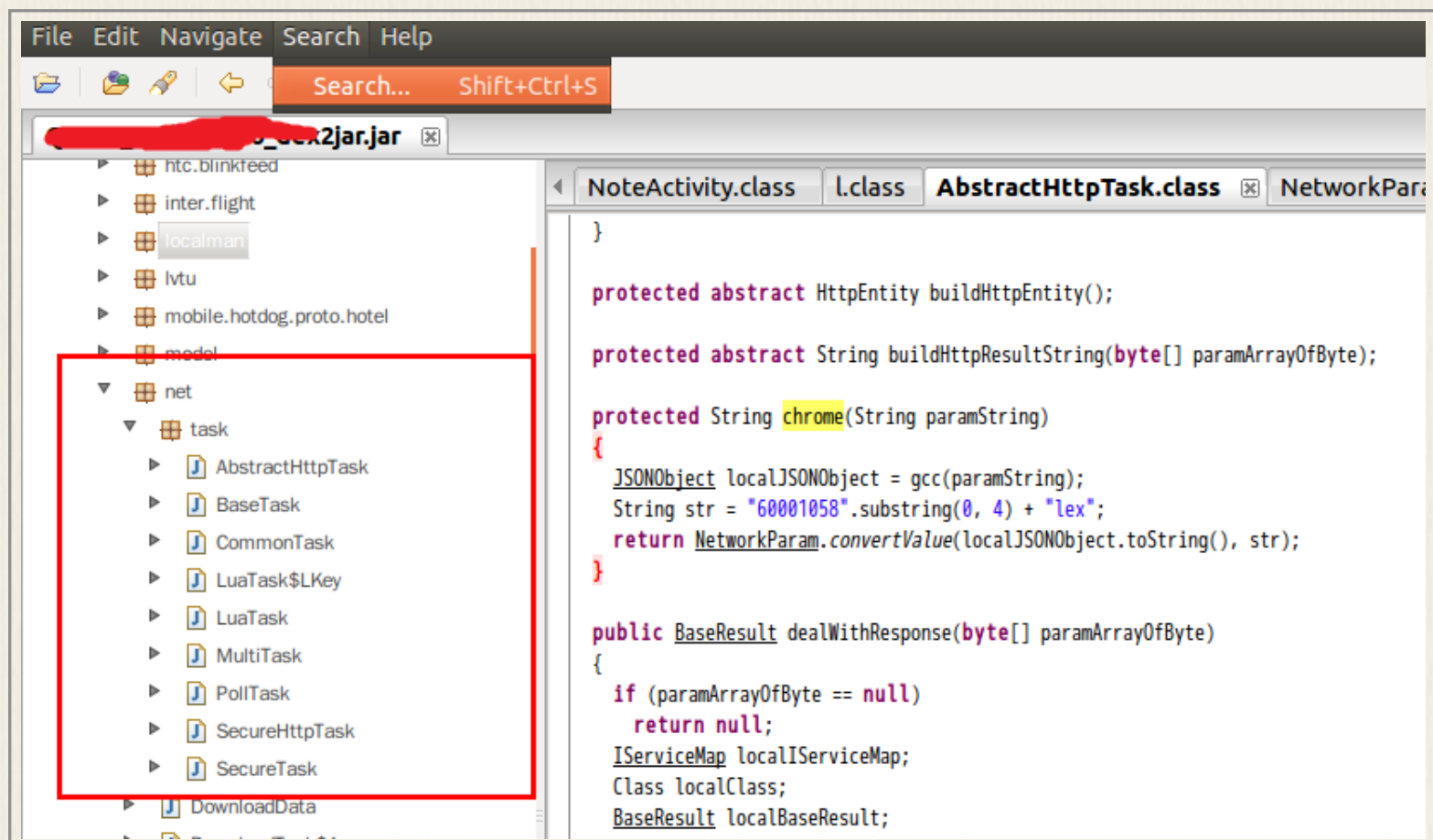
先来看onCreate中的操作，发现使用proguard对apk进行了混淆了。这种混淆虽然不会影响我们反编译出来的代码内容，但是由于对类名、函数名、变量名进行了随机命名，导致我们阅读代码的过程比较痛苦。

我的目的很明确，就是定位到处理提交数据的代码，没有必要去花大量的时间来阅读被混淆的代码，所以我决定使用动态跟踪程序运行的轨迹来定位我想要获得的代码。

2. 动态定位过程

虽然要用动态的方法来定位，但是还是需要简单的阅读java源码来确定提交数据的大概处理方式。

我的运气还是不错，网络传输部分的代码并没有被混淆。大体看了一下这些代码，发现和一般的app一样，客户端和服务端的数据交互也是使用json的格式进行的，并且使用了阿里开源的fastjson类来处理json内容。

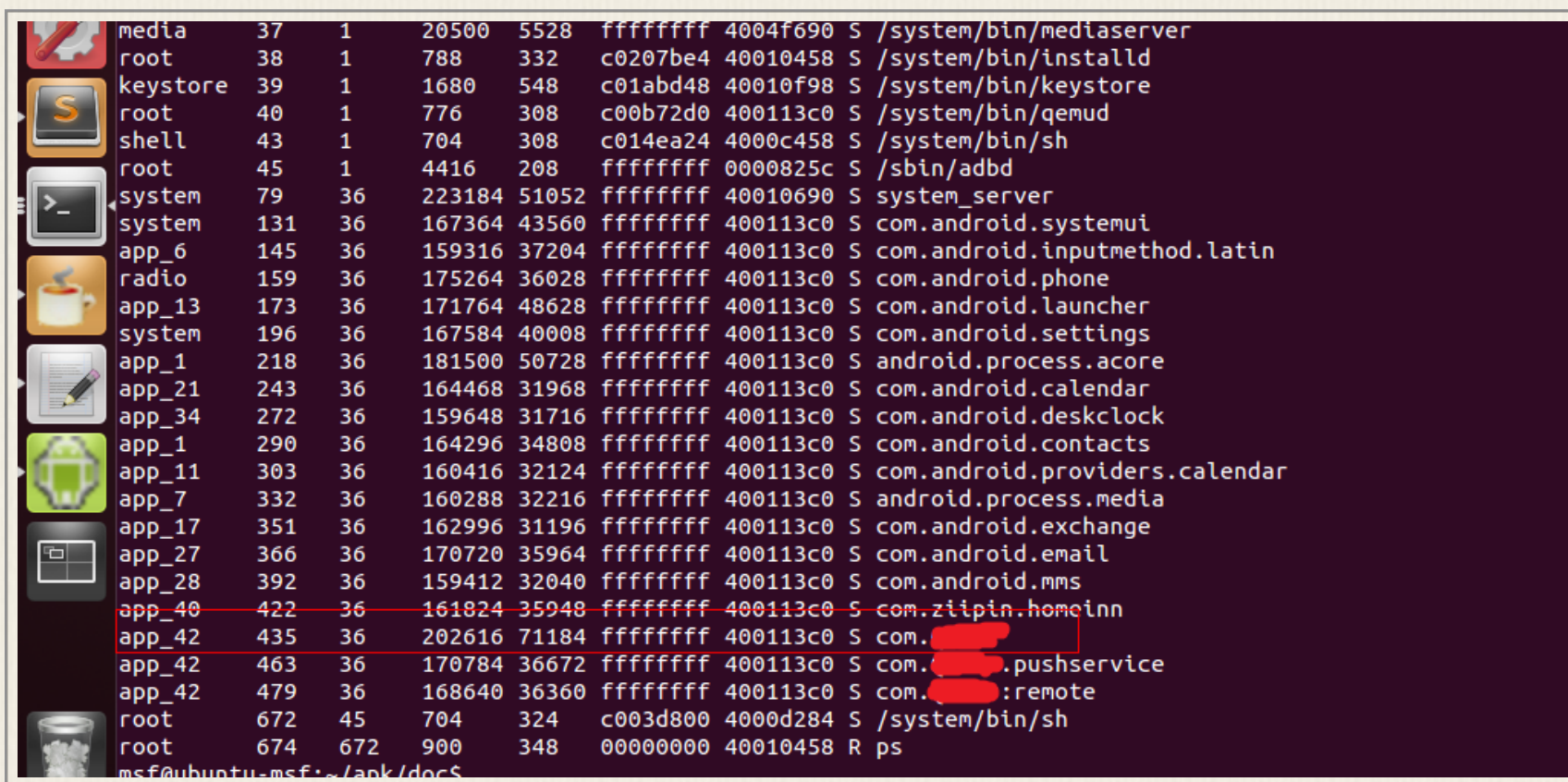


了解了以上的这些情况，我决定通过跟踪JSONObject这个类来定位处理提交数据的位置。

这里推荐一个分析app的神器——Andbug，虽然不能用来单步调试，但是动态跟踪app中各种线程调用栈、类调用栈、方法调用栈，断点获取当前内存中变量内容等功能还是非常实用的。

废话不多说了，来看操作吧，先获取要动态分析的app进程ID：

adb shell ps



media	37	1	20500	5528	ffffff	4004f690	S	/system/bin/mediasever
root	38	1	788	332	c0207be4	40010458	S	/system/bin/installd
keystore	39	1	1680	548	c01abd48	40010f98	S	/system/bin/keystore
root	40	1	776	308	c00b72d0	400113c0	S	/system/bin/qemu
shell	43	1	704	308	c014ea24	4000c458	S	/system/bin/sh
root	45	1	4416	208	ffffff	0000825c	S	/sbin/adbd
system	79	36	223184	51052	ffffff	40010690	S	system_server
system	131	36	167364	43560	ffffff	400113c0	S	com.android.systemui
app_6	145	36	159316	37204	ffffff	400113c0	S	com.android.inputmethod.latin
radio	159	36	175264	36028	ffffff	400113c0	S	com.android.phone
app_13	173	36	171764	48628	ffffff	400113c0	S	com.android.launcher
system	196	36	167584	40008	ffffff	400113c0	S	com.android.settings
app_1	218	36	181500	50728	ffffff	400113c0	S	android.process.acore
app_21	243	36	164468	31968	ffffff	400113c0	S	com.android.calendar
app_34	272	36	159648	31716	ffffff	400113c0	S	com.android.deskclock
app_1	290	36	164296	34808	ffffff	400113c0	S	com.android.contacts
app_11	303	36	160416	32124	ffffff	400113c0	S	com.android.providers.calendar
app_7	332	36	160288	32216	ffffff	400113c0	S	android.process.media
app_17	351	36	162996	31196	ffffff	400113c0	S	com.android.exchange
app_27	366	36	170720	35964	ffffff	400113c0	S	com.android.email
app_28	392	36	159412	32040	ffffff	400113c0	S	com.android.mms
app_40	422	36	161824	35948	ffffff	400113c0	S	com.zilpin.homeinn
app_42	435	36	202616	71184	ffffff	400113c0	S	com. .pushservice
app_42	463	36	170784	36672	ffffff	400113c0	S	com. :remote
app_42	479	36	168640	36360	ffffff	400113c0	S	/system/bin/sh
root	672	45	704	324	c003d800	4000d284	S	ps
root	674	672	900	348	00000000	40010458	R	

进程ID 445，使用Andbug挂载该进程，并使用classes命令查找fastjson类的全路径：

andbug shell -p 435 classes JSONObject

andbug shell -p435 classes JSONObject

```
## AndBug (C) 2011 Scott W. Dunlop <swdunlop@gmail.com>
>> classes JSONObject
## Loaded Classes
-- com.alibaba.fastjson.parser.deserializer.JSONObjectDeserializer
-- org.json.JSONObject
-- com.alibaba.fastjson.JSONObject
>>
```


这里提示一个使用classes和method命令查找的小技巧。我们在Andbug的shell环境下使用classes时很容易由于class 过多而导致没办法看到所有的class。这是我们可以终端环境下使用classes命令配合more来一点点的查看，就像这样：

```
andbug classes -p 435|more
```

然后我们使用class-trace命令来对这个类进行跟踪：

```
class-trace com.alibaba.fastjson.JSONObject
```

```
>> class-trace com.alibaba.fastjson.JSONObject
## Setting Hooks
-- Hooked com.alibaba.fastjson.JSONObject
>>
```

在app中随便触发一个会提交请求的事件，调用过程在终端中完美的呈现了出来：

```
## trace thread <13> AsyncTask #3 (running suspended)
-- com.alibaba.fastjson.JSONObject.put(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/Object;:0
-- this=Lcom/alibaba/fastjson/JSONObject; <830036424072>
-- com. net.task.AbstractHttpTask.gcc(Ljava/lang/String;)Lcom/alibaba/fastjson/JSONObject;:146
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.AbstractHttpTask.chrome(Ljava/lang/String;)Ljava/lang/String;:0
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.CommonTask.buildHttpEntity()Lorg/apache/http/HttpEntity;:73
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.AbstractHttpTask.getResult()Lcom/ model/response/BaseResult;:18
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.BaseTask.doInBackground([Ljava/lang/Void;)Lcom/ model/response/BaseResult;:92
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.BaseTask.doInBackground([Ljava/lang/Object;)Ljava/lang/Object;:2
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. utils.c.call()Ljava/lang/Object;:21
-- this=Lcom/ utils/c; <830039388176>
-- java.util.concurrent.FutureTask$Sync.innerRun():23
-- this=Ljava/util/concurrent/FutureTask$Sync; <830036564256>
-- java.util.concurrent.FutureTask.run():2
-- this=Lcom/Qunar/utils/d; <830039388200>
-- java.util.concurrent.ThreadPoolExecutor.runWorker(Ljava/util/concurrent/ThreadPoolExecutor$Worker;)V:28
-- this=Ljava/util/concurrent/ThreadPoolExecutor; <830028345088>
-- completedAbruptly=True
-- task=Lcom/Qunar/utils/d; <830039388200>
-- w=Ljava/util/concurrent/ThreadPoolExecutor$Worker; <830024737928>
-- java.util.concurrent.ThreadPoolExecutor$Worker.run():2
-- this=Ljava/util/concurrent/ThreadPoolExecutor$Worker; <830024737928>
-- java.lang.Thread.run():6
-- this=Ljava/lang/Thread; <830022442624>
## trace thread <13> AsyncTask #3 (running suspended)
-- com.alibaba.fastjson.JSONObject.put(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/Object;:0
-- this=Lcom/alibaba/fastjson/JSONObject; <830036424072>
-- com. net.task.AbstractHttpTask.gcc(Ljava/lang/String;)Lcom/alibaba/fastjson/JSONObject;:162
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.AbstractHttpTask.chrome(Ljava/lang/String;)Ljava/lang/String;:0
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.CommonTask.buildHttpEntity()Lorg/apache/http/HttpEntity;:73
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.AbstractHttpTask.getResult()Lcom/ model/response/BaseResult;:18
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.BaseTask.doInBackground([Ljava/lang/Void;)Lcom/ model/response/BaseResult;:92
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. net.task.BaseTask.doInBackground([Ljava/lang/Object;)Ljava/lang/Object;:2
-- this=Lcom/ net/task/CommonTask; <830039388112>
-- com. utils.c.call()Ljava/lang/Object;:21
-- this=Lcom/ utils/c; <830039388176>
-- java.util.concurrent.FutureTask$Sync.innerRun():23
-- this=Ljava/util/concurrent/FutureTask$Sync; <830036564256>
-- java.util.concurrent.FutureTask.run():2
-- this=Lcom/Qunar/utils/d; <830039388200>
-- java.util.concurrent.ThreadPoolExecutor.runWorker(Ljava/util/concurrent/ThreadPoolExecutor$Worker;)V:28
```


可以看到调用过程都用到了com.XXX.net.task.CommonTask中的方法，打开这个类的java源码第一眼就看到这段代码：

```
1  protectedHttpEntity buildHttpEntity()
2  {
3      if(this.hostUrl.indexOf("?")>0)
4          this.hostUrl=(this.hostUrl+"&qrt="+this.networkTask.param.key.getDesc())
5      while(true)
6      {
7          Stringstr=String.valueOf(this.networkTask.param.ke);
8          StringBuilderlocalStringBuilder1=newStringBuilder();
9          localStringBuilder1.append("c="+chrome(str));
10         localStringBuilder1.append("&");
11         StringBuilderlocalStringBuilder2=newStringBuilder("b=");
12         BaseParamlocalBaseParam=this.networkTask.param.param;
13         SerializerFeature[]arrayOfSerializerFeature=newSerializerFeature[1];
14         arrayOfSerializerFeature[0]=SerializerFeature.WriteTabAsSpecial;
15         localStringBuilder1.append(NetworkParam.convertValue(JSON.toJSONString(l
16         if((this.networkTask.param.param instanceofHotelBookParam))
17         {
18             HotelBookParamlocalHotelBookParam=(HotelBookParam)this.networkTask.par
19             if(localHotelBookParam.vouchParam!=null)
20                 dealVouchRequest(localStringBuilder1,localHotelBookParam.vouchParam)
21         }
22         localStringBuilder1.append("&");
23         localStringBuilder1.append("ext="+NetworkParam.convertValue(XXXApp.getCo
24         localStringBuilder1.append("&v=alex");
25         this.networkTask.param.url=localStringBuilder1.toString();
26         try
27         {
28             StringEntitylocalStringEntity=newStringEntity(this.networkTask.param.u
29             returnlocalStringEntity;
30             this.hostUrl=(this.hostUrl+"?qrt="+this.networkTask.param.key.getDesc(
31         }
32         catch(UnsupportedEncodingExceptionlocalUnsupportedEncodingException)
33         {
34             cl.m();
35         }
36     }
37     returnnull;
38 }
```

结合之前的抓包，这应该就是我要找的地方了。从中找到处理c参数的代码，看到调用了com.XXX.net.task.AbstractHttpTask.chrome对参数值进行了处理，跟进chrome方法：

```
1  protectedStringchrome(StringparamString)
2  {
3      JSONObjectlocalJSONObject=gcc(paramString);
4      Stringstr="60001058".substring(0,4)+"lex";
5      returnNetworkParam.convertValue(localJSONObject.toString(),str);
6  }
```

继续赶进到convertValue方法：

```
1 publicstaticStringconvertValue(StringparamString1,StringparamString2)
2 {
3     if(TextUtils.isEmpty(paramString1))
4         return "";
5     if(paramString2==null)
6         paramString2="";
7     try
8     {
9         Stringstr=URLEncoder.encode(Goblin.e(paramString1,paramString2),"utf-8")
10         returnstr;
11     }
12     catch(ThrowablelocalThrowable)
13     {
14         localThrowable.printStackTrace();
15     }
16     return "";
17 }
```

感觉的胜利的曙光越来越近了，这个Goblin.e应该就是最后的加密方法了吧，谁知打开这个文件（内心一万只草泥马在狂奔）：

```
1 packageXXX.lego.utils;
2 importcom.XXX.XXXApp;
3
4 publicclassGoblin
5 {
6     static
7     {
8         try
9         {
10             System.loadLibrary("goblin_2_5");
11             return;
12         }
13         catch(UnsatisfiedLinkErrorlocalUnsatisfiedLinkError1)
14         {
15             try
16             {
17                 System.load("/data/data/"+XXXApp.getContext().getPackageName()+
18 /lib/lib"+"goblin_2_5"+"so");
19                 return;
20             }
21             catch(UnsatisfiedLinkErrorlocalUnsatisfiedLinkError2)
22             {
23             }
24         }
25     }
26 }
```

```

27     publicstaticnativeStringSHR();
28
29     publicstaticnativeStringd(StringparamString1,StringparamString2);
30
31     publicstaticnativeStringdPoll(StringparamString);
32
33     publicstaticnativeStringda(StringparamString);
34
35     publicstaticnativeStringdn(byte[]paramArrayOfByte,StringparamString);
36
37     publicstaticnativebyte[]dn1(byte[]paramArrayOfByte,StringparamString);
38
39     publicstaticnativeStringduch(StringparamString);
40
41     publicstaticnativeStringe(StringparamString1,StringparamString2);
42
43     publicstaticnativeStringePoll(StringparamString);
44
45     publicstaticnativeStringea(StringparamString);
46
47     publicstaticnativebyte[]leg(byte[]paramArrayOfByte);
48
49     publicstaticnativeStringes(StringparamString);
50
51     publicstaticnativeintgetCrc32(StringparamString);
52
53     publicstaticnativeStringgetPayKey();
54
55     publicstaticnativeStringve(StringparamString);
    }

```

3.调用so文件函数

居然把加密方法写到了so文件中！难道要去看ARM汇编？

既然这个so文件中有加密函数，那是不是就应该有解密函数，那我应该还是可以偷懒的吧。

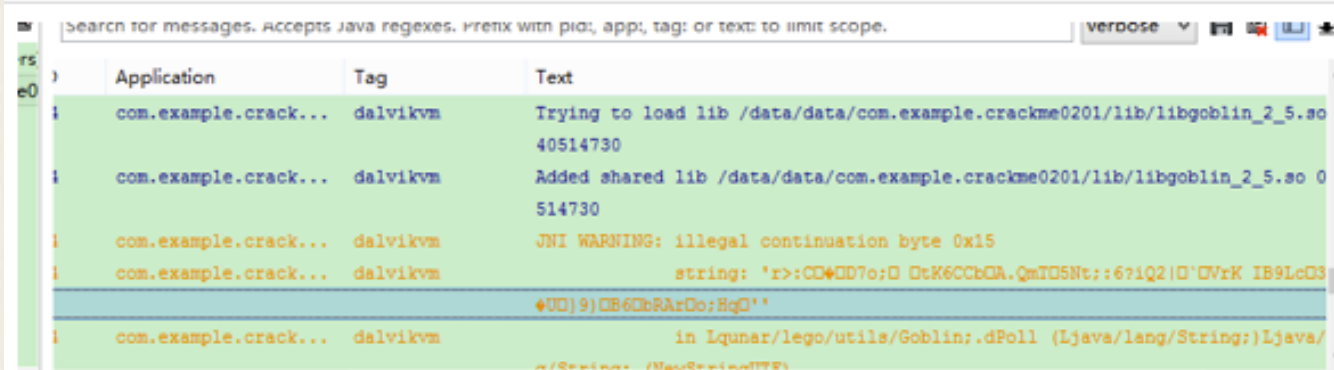
我们在上面看到的e函数肯定是用来加密的，那那个d函数是不是用来解密的（encode和decode）？

自己本地创建一个app，并且创建一个XXX.lego.utils包，添加一个Goblin.java文件，把我们刚刚看到的Goblin源码粘贴进去。然后在app的一个Activity中导入Goblin，并在OnCreate中调用d函数来尝试解密。部分代码如下：


```

#! java
Stringc="B946D7CF7B9E5B589F8DE6BC9E5DACF08DA6B35DA7A899DCA5AD5A58ADDAD5E6
6363589EF2D385B1ACACABDAD5E65F63589EF2D3F5B43EA7ACE36DFCA5383EABE7D4F1403
E379FF0DEFCAD9FABA7E6E1F243A7AAAC74E380A4A09E6593D3FEA4ABA8ACF0DDF0AEAAB3
A7EAE9FBA4A09E5F97D3FEA49EA09E9B97A9A4B69EAD7E1F6B0AC9EA0DA94A95E57609EF
2D3B55E659EA0DA94B55F9EB69EDAD5E668689EB6DA98AA6E576E62939DE6A69E616D868C
B673636162DAEBE6AEA2ACA2E486F3AD9EA09EA898A0A4B69EABE8E1F3B29EA09EA998A0A
4B69EAD7E3E385AE3DA7ABDB6FF4B93A9F3DEF3FBA537ACAD77D486AD37ADABE77383B4B4
ACB4DAD5E66E9EB69EA886AF634061599F97E6A69E676394D3FEA4ACACACE8E1F4B2ACACA
CE8E1F4B2AC9EA0DA9CAAA4B69E9EDCD3B269589EB6DADFF4B2ACABACE3E9E675";
Stringp2="6000lex";
Stringtest=Goblin.d(c,p2);
System.out.println(test);

```



天不遂人愿啊！解密出错，看来真的要去看ARM汇编了。。。。。。

4.动态调试so文件

由于app自带的加密数据，我们不知道原来的样子，所以要自己构造一个字符串加密，来调试。修改上面的app代码如下：

```
#! java
```

```
Stringjson="json{"/"test"/:"/"test1"/,"/"test4"/:"/"test1"/,"/"test5"/:"/"test1"/,"/"test6"/:"/"test1"/,"/"test3"/:"/"test1"/,"/"test2"/:"/"test1"/,"/"test1"/:"/"test1"/}";
```

```
Stringp2="6000lex";
```

```
Stringtest=Goblin.e(c,p2);
```

```
System.out.println(test);
```

生成代码如下：

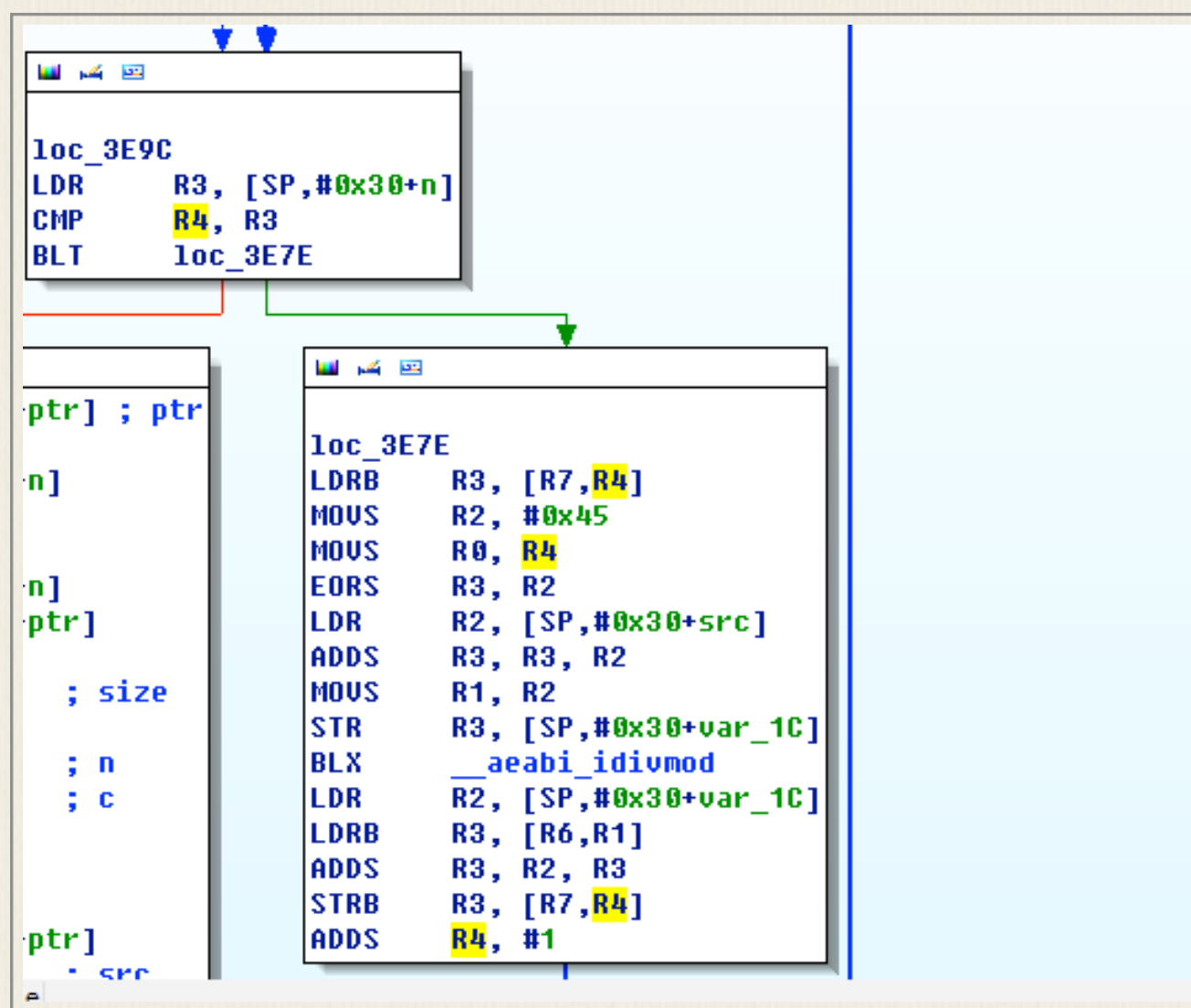
C0990850B69C969E92C19C9DC5CDD9F0FAB99AD3960CE3F6D2C
FB0AA9AE904F6C0A099A68DFFD0C1EE978CA985CAD2FCF6CD92A7
C4FCE106CCC99CC39C11F7F3D0A9BDAD8AE3E0D9C3BC898EB8DC
D3F2BB8B8BB3C010D9DAFAB99AD3960FE30CD2CFB0AA9AEC04E0
C0A099A68DFFD0D7EE978CA985CED2B5

好了准备活动完成了，下面我们开始动态跟踪之旅吧。在《Android软件安全与逆向分析》中提供的动态分析工具是IDA pro 6.1以上版本，这个我在调试过程中发现加载很慢。虽然加载完成后，能够跟着IDA生成的流程图来调试很爽，但是加载成功率实在是太低了。所以，我放弃了用IDA进行动态调试，而是选择了这个号称移动端Onlydbg的gikdbg来进行调试，同时配合IDA的流程图。

gikdbg使用参考《gikdbg.art系列教程2.1-调试so动态库》这篇blog很容易上手，这里也就不多说了。

调试跟踪过程很枯燥，也没什么可以说的，我们直接看结过吧。

通过反复的动态跟踪，确定下面这个循环是加密的关键：



可以看出加密方法比较简单，对于源数据的每一个字符与0x45进行异或，然后jia0x24，最后再加上硬编码在so文件的一串key中的一个字符。根据汇编逆向出来的python代码如下：

```
result=""
i=0
while(i<len(json)):
    char=json[i]^69
    j=i
    if j>len(key):
        j=j%len(key)
    encode=int(ord(char))+36+key[j]
    result+=encode
    i+=1
```

在加密完数据后，会在数据头部添加一个8个字符（32位）的校验数据，校验算法使用的是adler32。由此可以推出解密算法，代码如下：

```
def decode():
```

```
    ejson='B69C969E92C19C9DC5CDD9F0FAB99AD3960CE3F6D2CFB0
AA9AE904F6C0A099A68DFFD0C1EE978CA985CAD2FCF6CD92A7C4F
CE106CCC99CC39C11F7F3D0A9BDAD8AE3E0D9C3BC898EB8DCD3F
2BB8B8BB3C010D9DAFAB99AD3960FE30CD2CFB0AA9AEC04E0C0A0
99A68DFFD0D7EE978CA985CED2B5'
```

```
    key='cBHO06GYkxNModVyAtXiGzIPETyS5KUL8gE4'
```

```
    i=0
```

```
    result=""
```

```
    while(i<len(ejson)):
```



```

j=i/2
char=ejson[i:i+2]
ifj<len(key):
    k=key[j]
else:
    k=key[j%len(key)]
i=i+2
c=int(char,16)-int(ord(k))
ifc<0:
    c+=128
c=c-36
ifc<0:
    c+=128
c=c^69
dchar=chr(c)
result+=dchar

printresult

decode()

```

其中ejson中的内容为，我们使用e函数加密后获得的内容剔除前八位，解密效果如下：

0×02 最终结果&分析总结

不过悲剧的是用这个解密方法没办法解密前面我抓包获取的数据。。。。。

郁闷之心无以言表啊！！！！

不过这个过程还是很有意义的，了解了Android各种姿势的动态调试方法。这里再次回顾一些这个过程。

首先通过反编译获取smali和java代码进行静态分析，发现代码被混淆后，明确自己的最终目标——找到处理提交请求的方法，然后进行动态跟踪。动态跟踪和静态分析结合定位出处理提交请求的几个类，翻看这些类的代码，来找到最终我们想找的方法。

在发现处理方法使用了so文件中的函数，通过自己构造app来分别调用so中的各个函数，试图从中找到直接的解密函数。

在so中没有找到解密函数的情况下，通过动态调试与静态查看汇编，分析出加密算法，并写出解密工具。

原文链接：<http://drops.wooyun.org/tips/2871>

Whitepages的架构变迁：从Ruby到响应性更好的Scala和Akka

作者：臧秀涛

Whitepages是位于美国的一家公司，主要负责提供个人和企业的联系信息，供用户搜索。其业务每个月要服务5000万独立用户，每天要完成3500万次搜索。其移动产品每个月也有超过1800万的活跃用户。

随着业务的增长，Whitepages的架构出现了瓶颈。经过评估，开发人员将出现瓶颈及代价较高的部分从原来的Ruby语言实现迁移到了更为现代、响应性更好的Scala语言和Akka框架。Whitepages的开发人员John Nestor和Dragos Manolescu分享了他们的经验。

在介绍了公司要应对的业务规模之后，他们提到了Ruby遗留系统存在的问题：

- 较高的延迟
- 较高的资源消耗，包括内存和处理器两个方面
- 对于上游服务的降级支持较差
 - 并发能力有限
 - 当阻塞在较慢的上游服务上时，工作线程会饥饿
 - 连接管理和恢复能力不佳

之所以选择Scala，是因为这门语言具有如下优点：

- 优雅地结合了函数式编程范型和面向对象编程范型
- 静态类型系统
 - 类型推导可以避免编写大量的Java样板代码
 - 编译器可以捕获很多错误

- 运行在JVM上
 - 速度快
 - 几乎可以无缝地与JVM库互操作
 - 相当成熟的工具支持
- 基于Actor的并发框架——Akka

Whitepages的响应式服务的特点：

- 面向服务的架构：通信采用Thrift或HTTP上的Json
- 延迟和吞吐量非常重要
- 对日志和监控有很高的要求
- 敏捷的开发、测试、构建和部署流程

在使用Scala和Akka迁移了服务之后，改进非常明显。

Service	p50 ms	p99 ms	through RPS/core
DirSvc - Scala	25	300	80
DirSvc - Ruby	140	1200	7

他们先从1个单一的后台服务入手，现在已经完成了4个服务的迁移；还有6个服务尚在开发之中。Scala开发人员也从最初的6个增加到20个。未来他们还将迁移更多服务。

他们总结的成功经验主要有以下几点：

- Scala简洁的语法提高了开发效率。
- 异步代码提高了性能。
- 不可变集合和函数式编程减少了bug。

- 强类型检查也有助于减少bug，并使代码的可维护性更好（不过元编程变困难了）。
- 并发能力提高。
- Spray具有极好的性能，而且提供了一个异步API。
- SBT能够根据需求轻松定制，尽管学习曲线有些陡峭。
- IntelliJ IDEA对Scala的支持非常好。
- Typesafe的开发者支持合约非常不错，Typesafe反馈非常快，对复杂的问题也可以给出很好的答案。

当然，迁移过程中也遇到了不少问题，比如：

- 差劲的文档，SBT就是个典型，很多时候还不得不阅读Scala和Akka库的源代码。
- API不稳定，升级步子太大。
- 缺乏好用的并发构件分析工具：尝试过Typesafe Console，但是一直没有完整地跑起来，最后放弃；虽然有些新工具，但没有时间一一评测。
- 生态系统不如Java，缺乏一些所需的组件；有时候选择太多，比如Json库就有10多款；GitHub上存在大量的Scala项目，但质量参差不齐。
- 难以调试，尤其是异步代码和Actor。
- 语言和库的问题：类型擦除是一个主要缺陷；Actor缺乏类型检查；某些Scala代码看上去简单直观，但是要了解其背后的机制也非常困难。

不过整体而言还是利大于弊，Scala/Akka非常适合构建响应式系统。

最后，他们讲到了开发人员这个关键因素。有经验的Scala开发人员还不够多。所以他们一方面招聘Scala开发人员，一方面培训现有的Ruby开发人员，促其转型。

Whitepages并不是第一家尝试从Ruby向其他开发语言迁移的公司。Twitter早在2011年就开始从Ruby向Scala和Java迁移。Iron.io从Ruby迁移

到Go，服务器从30台减少到2台。LinkedIn从Rails迁移到Node，服务器减少了27台，速度提升高达20倍。

原文链接：<http://www.infoq.com/cn/news/2014/08/whitepages>

美团云的技术演变：先把云主机做稳定了再说别的

作者：唐君毅 邱剑 朱宴

2013年上半年，美团发布了其公有云服务美团云。该产品一开始的背后支撑团队是美团系统运维组，到2014年6月，美团云业务部从系统运维组独立了出来，专门负责云计算方面的产品研发与运营。

近日，InfoQ中文站编辑与美团云业务部的三位工程师进行了交流，了解美团云目前的状态、过去的演变历程和下一步发展计划。以下内容根据这次交流整理而成。

背景

独立出来的美团云业务部目前有十几位工程师。虽然部门独立，但工作中仍然跟系统运维组有紧密的配合。美团系统运维组原本就有三分之二的开发工程师，而运维工程师也全部具备编写代码的能力，因此开发与运维工程师能够进行紧密的配合。

美团云的最初版本起步于2012年7月，一开始是作为私有云计算平台来构建。第一版开发了大约2个月的时间，之后用了大概10个月的时间将美团的所有业务迁移到云平台上。现在除了Hadoop、数据库仍跑在物理机上之外，美团网的所有业务都已经运行在美团云上，内部的研发、测试平台也运行在美团内部的办公云平台上。

2013年5月，美团云开始对外提供公有云服务，截止到目前仅对外提供云主机产品。对象存储、Redis、MySQL、负载均衡、监控、VPC等服务也在研发，部分产品已经在内部以及部分测试客户使用，未来会根据产品的成熟度和客户的需求程度逐步对外开放。

稳定是公有云服务的核心价值。自从公有云服务开放以来，美团云主要的工作都放在提升云主机的稳定性，完善云主机模板、备份、监控、安全等工作上。预计在2014年9月，美团云将发布其第三个版本的更新，继续打磨其产品细节。

技术架构

美团云最初选型的时候对OpenStack、CloudStack、Eucalyptus都做过调研，调研的结果是架构设计使用OpenStack的框架，网络架构参考CloudStack，主要组件由自己开发，部分组件在OpenStack原生组件上进行了二次开发。

- 核心的云主机管理系统是自己研发，没有使用Nova。采用Region-Zone-Cluster三层架构，支持跨地域、多数据中心的大规模集群部署。采用了基于KVM的主机虚拟化和基于OpenVSwitch+OpenFlow的网络虚拟化技术。

- 镜像管理用了Glance。有一定修改，例如，多数据中心分布式支持，以及镜像替换。

- 身份管理用了Keystone。有一定修改，例如，高并发性能改进，与美团帐户系统的集成。

- 对象存储用了Swift，但Swift在写延迟方面存在性能问题，同时在OAM方面的功能比较薄弱，所以也做了一些修改和研发。Swift现在已经在为美团网内部存储量几十TB量级的业务提供服务。

之所以没有整个引入OpenStack，是因为当时调研时，感觉OpenStack的设计比较脱离美团的实际情况。例如，网络架构需要有较大的调整，同时需要有共享存储的支持。当时对美团来说，现有业务的基础设施已经基本固化，为适应OpenStack而做这样的调整基本不可接受。另外，OpenStack在很多细节方面达不到需要的级别。比如，OpenStack对跨机房多Zones的设计，它假设你机房之间的网络是完备的，这也不太符合我们的网络现状。因此，我们基于美团现有的主机使用模式和网络架构，重新设计了网络、存储和主机管理模型，自主研发了虚拟化管理平台。同时，我们在从单机房做到多机房的时候，在Zones之间做了解耦，比如在每个机房

里都放置Glance服务节点，减少对跨机房网络的依赖。正是由于这些研发工作，使得我们经过不到一年时间的磨练，就实现了美团整个基础设施完全运行在私有云上的目标。

当然，由于我们不使用Nova，就意味着OpenStack社区的很多依赖Nova的组件和功能我们基本无法直接使用。不过，相对于OpenStack大而全/兼容并包的架构，我们坚持在每一方面都集中精力用好一种技术，如主机虚拟化使用KVM，网络虚拟化使用OpenVSwitch+OpenFlow，使得整个系统的开发和维护成本相对较低，同时能深挖这些技术方案的功能特性，最大限度地压榨硬件性能。同时，由于我们掌握了基础系统的代码，使得我们可以较高的效率添加一些新的业务功能（例如，对虚拟IP，USB KEY的支持等），以及实现系统架构的升级改造（例如，对多机房架构支持等）。另外，我们对使用的OpenStack组件做的一些修改，比如对Swift的优化，目前技术委员会也在提议如何回馈给上游社区。当然了，这个还需要看社区对我们的patch接受与否，而我们也还是以满足业务需求优先。

其他方面，块存储落在本地的SAS盘上并在本地做RAID。目前我们对美团网自己的业务做RAID5，对公有云用户做RAID10。这是考虑到美团网自己的业务在应用层已经做了较完备的高可用设计的，即使掉了单个节点也不会影响到业务；但对于公有云用户而言，他们用的那一台云主机就是一个单点，所以要对他们的云主机做更好的保护。使用RAID10当然成本会比较高。我们也在考虑共享存储，当然前提是先解决了上面的稳定性和性能问题。以后会使用SSD，使块存储的性能有更大的提升。

网络方面做了分布式设计，主机上用了OpenFlow，通过OpenFlow修改二层协议，让每个用户拥有一个独立的扁平网络，跟其他用户的网络隔离。通过DNS虚拟化技术，使得不同的用户可以在各自的私有网络上使用相同的主机名字，并在每个宿主机上部署分布式DNS和DHCP以实现基础网络服务的去中心化。

运维

美团云的运维思路跟整个美团的运维思路是一致的。下面介绍的运维思路既适用于美团云，也适用于整个美团网。

运维框架可以概括为五横三纵。从横向来看，自底向上分为五个层次：

- 物理层，包括机房网络、硬件设施。我们已在开展多机房和城域网建设，从最底层保证基础设施的稳定性。为了应对大规模机房建设带来的运维成本，我们实现了Baremetal自动安装部署的Web化管理，从服务器上架之后，其他工作均由自动化完成，并可以和虚拟机一样管理物理机。
- 系统层，包括操作系统、虚拟化。我们在虚拟化基础之上采用了模板化（镜像）的方式进行管理，也对Linux内核做了一部分定制开发，例如针对OVS的兼容性做了优化。
- 服务层，包括Webserver、缓存、数据库等基础服务。我们基于Puppet工具做了统一配置管理，有自己的软件仓库，并对一部分软件包做了定制。统一配置管理的好处，一方面是避免不一致的修改，保证集群的稳定性，另一方面是提高运维效率。
- 逻辑层，包括业务逻辑、数据流。这一层的主要工作是发布和变更。在很多其他公司，业务的发布上线、数据库的变更管理都是由运维来做，我们认为这样对开发、运维的协作成本较高，所以一直往开发人员自助的方向做，通过代码发布平台、数据库变更平台实现开发和运维工作的轻耦合。在发布平台中，每个应用对应独立的集群，有一位开发作为应用owner有最高权限，有多位开发作为应用的成员可以自助发布代码。数据库变更平台也有类似的权限控制机制，并在任务执行层面有特殊的稳定性考虑，例如将大的变更任务自动调度到夜间执行，对删除数据表的任务在后台先做备份。
- 应用层，包括用户可见部分。除了跟逻辑层有类似的发布和变更之外，我们有统一前端平台，实现访问流量的进出分离、行为监测和访问控制，这对于整体的安全性有很大的好处。

从纵向来看，有三部分工作，对上述五个层次是通用的：

- 监控。从物理层到服务层的监控和报警都是运维来跟进、响应的。对于逻辑层和应用层，也是开发人员自助的思路，运维提供监控API的规范，开发可以自己创建监控项、设定报警规则、进行增删改查。监控报警之后的处理，现在有些做到了自动化，有些还没有。尤其是有些基础架构和业务之间的纵向链条还没有打通，包括建立业务容量模型，某种特定的业

务形态在多少用户的情况下最高负载多少，不同负载等级下的SLA应该有多少，等等，这些模型都建立起来之后就能够进行自动化的处理。

- 安全。我们很早就部署了统一的安全接入平台，所有线上的人工操作都需要登陆relay跳板机，每个人有独立的登陆帐号，所有线上操作都有审计日志。更多的安全工作由专门的信息安全组负责。

- 流程。早期基于Jira做了一些简单的流程，但仍需要改进。现在正在针对比较集中的需求，开发相应的流程控制系统，方向也是自动化、自助化。从业务部门申请VM资源，到业务扩容的整个流程，我们正在进行上下游的打通，未来可以在Web界面上通过很简单的操作实现，也提供服务化的API，方便其他业务平台进行集成。虚拟化覆盖全业务线之后，这些事情做起来都变得很方便。

总之，美团网整体的运维思路就是：保证业务稳定运行，同时推动全面自动化、自助化。涉及开发、运维沟通协作的部分，尽可能通过自动化平台的方式，由开发人员自助完成。运维人员除了基础环境、平台建设之外，帮助业务进行高可用架构的梳理，提高代码的可运维性，以及定位和解决业务中的各类问题。

改进与演变

美团云从对内服务开始到现在两年以来，最大的一次改进就是从单机房到多机房的建设，这是跟美团网的城域网建设同步开展的。

单机房的时候，美团网业务早期曾遇到过运营商网络中断几小时的情况，期间业务不可用，非常痛苦。多机房冗余做到最理想的情况下是，即使一个机房整个断电了，业务也不受影响，当然这就意味着需要100%的冗余量，成本是比较高的。不过对于美团网来说，冗余的成本是很愿意承担的，因为业务不可用造成的损失要大于做这些冗余的成本，所以我们现在物理资源都留有50%的冗余，带宽一般会预留30%的冗余。

因为美团网的发展速度很快，去年我们一度遇到资源不够用的情况，在这上面踩了很多坑之后，开始做一些长远规划。现在美团网业务的双机房冗余已经实施了一部分，美团云也有两个机房，如果公有云客户的业务支持横向扩展，那么也可以做跨机房部署。这种机房级高可用做好了，对稳定性

又是一个很大的提升，大大减少网络抖动对业务的影响，可用性SLA可以从现在的4个9做到更高。有些规模比较大的客户对服务质量会有比较高的需求，所以美团的城域网、以及未来的广域网，也会共享给我们的公有云客户。

另外上面说到我们数据库跑在物理机上，这一块现在用的是SSD，读写性能顶得上早期的三台15000转SAS，瓶颈在千兆网卡上，所以我们现在也在做万兆网络的升级改造。数据库服务以后也会开放给公有云用户使用，基础设施跟美团自身业务一致。

未来的计划

由于使用本地存储，所以现在虚拟机迁移需要在夜间进行，以减少对用户服务的影响。为了提高服务的可用性，在确保稳定性和性能的前提下，共享存储是一个不错的选择，所以我们正在测试万兆网络下的共享存储方案。另外，我们底层虚拟化机制用的KVM，本身是没有热插拔的功能，这也是我们计划要做的一件事。

现在很多客户问我们，什么时候出Redis，什么时候出云数据库，一些客户对Redis和MongoDB会有需求，Web服务想要MySQL。我们的计划是由DBA团队提供一些模板，相当于是一些专门针对Redis/MySQL做好优化的系统镜像，让客户可以直接拿来用。这可能会在下一个版本 release 的时候推出。

我们还会提供一些基础架构的咨询服务，这个咨询服务一方面是工程师提供的人工服务，另一方面是以工具+文档的形式，以互联网的方式将我们的最佳实践共享出去。美团网做到现在的几百亿规模，内部有很多经验积累，如果能把这些积累传递给我们的客户，能够帮助客户少走很多弯路。

分享者简介

唐君毅，美团云产品工程师。哈工大毕业，曾任积木恒硕产品总监。现负责美团云的产品研发和运营工作。

邱剑，美团云架构师。清华毕业，曾任安和创新副总经理。现负责美团云的平台研发和架构工作。

朱晏，美团网高级技术经理。清华、中科院计算所毕业，曾任百货网联合创始人。现负责美团网的系统运维、云计算工作。

原文链接：<http://www.infoq.com/cn/articles/meituan-cloud-technique-evolution>

基于Storm的Nginx log实时监控系统

作者：钟国英

背景

UAE(UC App Engine)是一个UC内部的PaaS平台，总体架构有点类似CloudFoundry，包括：

1. 快速部署：支持Node.js、Play!、PHP等框架
2. 信息透明：运维过程、系统状态、业务状况
3. 灰度试错：IP灰度、地域灰度
4. 基础服务：key-value存储、MySQL高可用、图片平台等

这里它不是主角，不作详细介绍。

有数百个Web应用运行在UAE上，所有的请求都会经过UAE的路由，每天的Nginx access log大小是TB级，如何实时监控每个业务的访问趋势、广告数据、页面耗时、访问质量、自定义报表和异常报警？

Hadoop可以满足统计需求，但秒级的实时性不能满足；用Spark Streaming又有些大材小用，同时我们也没有Spark的工程经验；自写分布式程序调度比较麻烦并且要考虑扩展、消息流动；

最后我们的技术选型定为Storm：相对轻量、灵活、消息传递方便、扩展灵活。

另外，而由于UC的各地集群比较多，跨集群日志传输也会是其中一个比较大的问题。

技术准备

基数计数(Cardinality Counting)

在大数据分布式计算的时候，PV(Page View)可以很方便相加合并，但UV(Unique Visitor)不能。

分布式计算的情况下，几百个业务、数十万URL同时统计UV，如果还要分时段统计(每分钟/每5分钟合并/每小时合并/每天合并)，内存的消耗是不可接受的。

这个时候，概率的力量就体现了出来。我们在Probabilistic Data Structures for Web Analytics and Data Mining可以看到，精确的哈希表统计UV和基数计数的内存比较，并不是一个数量级的。基数计数可以让你实现UV的合并，内存消耗极小，并且误差完全在可接受范围内。

可以先了解LogLog Counting，理解均匀哈希方法的前提下，粗糙估计的来由即可，后面的公式推导可以跳过。

具体算法是Adaptive Counting，使用的计算库是stream-2.7.0.jar。

实时日志传输

实时计算必须依赖于秒级的实时日志传输，附加的好处是可以避免阶段性传输引起的网络拥堵。

实时日志传输是UAE已有的轻量级的日志传输工具，成熟稳定，直接拿来用了，包括客户端(mca)和服务端(mcs)。

客户端监听各个集群的日志文件的变化，传输到指定的Storm集群的各台机器上，存储为普通日志文件。

我们调整了传输策略，使得每台Storm机器上的日志文件大小大致相同，所以Spout只读取本机数据即可。

数据源队列

我们并没有用Storm常用的队列，如Kafka、MetaQ等，主要是太重了...

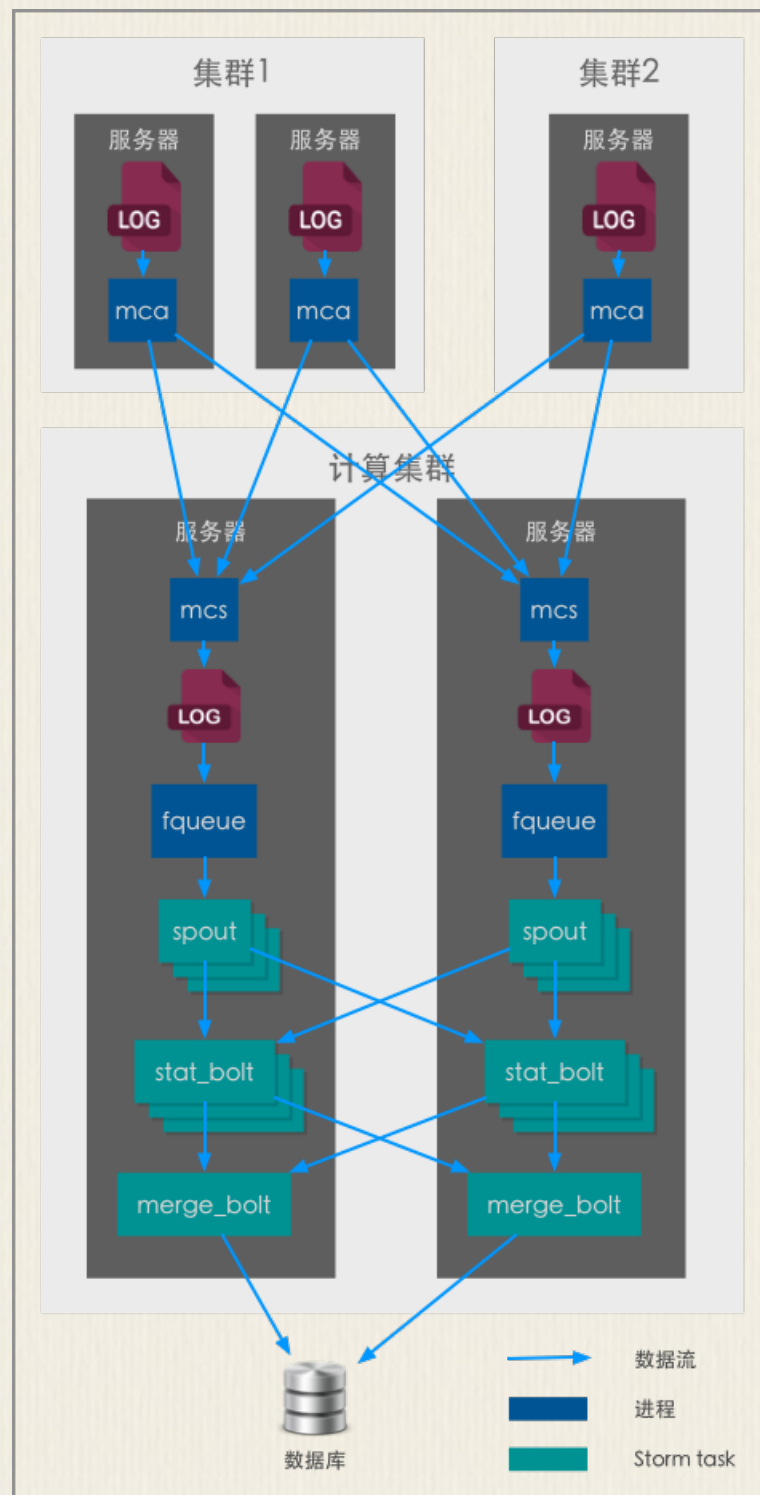
fqueue是一个轻量的memcached协议队列，把普通的日志文件转为memcached的服务，这样Storm的Spout就可以直接以memcached协议逐条读取。

这个数据源比较简单，它不支持重新发射(replay)，一条记录被取出之后就不复存在，如果某个tuple处理失败或超时，则数据丢失。

它比较轻量，基于本地文件读取，做了一层薄的缓存，并不是一个纯内存的队列，它的性能瓶颈在于磁盘IO，每秒吞吐量跟磁盘读取速度是一致的。但对于我们这个系统已经足够，后续有计划改成纯内存队列。

架构

通过上面的技术储备，我们可以在用户访问几秒后就能获取到用户的日志。



整体架构也比较简单，之所以有两种计算bolt，是基于计算的均匀分布考虑。业务的量相差极大，如果仅按业务ID去进行fieldsGrouping，计算资源也会不均衡。

1. spout将每条原始日志标准化，按照URL分组(fieldsGrouping，为保持每台服务器计算量的均匀)，派发到对应的stat_bolt上；

2. stat_bolt是主要的计算Bolt，将每个业务的URL梳理并计算，如PV、UV、总响应时间、后端响应时间、HTTP状态码统计、URL排序、流量统计等；

3. merge_bolt将每个业务的数据合并，如PV数，UV数等。当然，这里的UV合并就用到了前面提到的基数计数；

4. 自写了一个简单的Coordinator协调类，streamId标记为"coordinator"，作用：时间协调(切分batch)、检查任务完成度、超时处理。原理跟Storm自带的Transactional Topology类似。

5. 实现一个Scheduler通过API获取参数，动态调整Spout、Bolt在各服务器的分布，以便灵活分配服务器资源。

6. 支持平滑升级Topology：当一个Topology升级的时候，新Topology和旧Topology同时运行，协调切换时间，当新的Topology接管了fqueue之后，过河拆桥，杀死旧的Topology。

注意点：

- Storm机器尽量部署在同一个机柜内，不影响集群内的带宽；
- 我们的Nginx日志是按小时切分的，如果切分的时间不准确，在00分的时候，就可以看到明显的波动，所以，尽量使用Nginx module去切日志，用crontab发信号切会有延迟。切日志这种10秒级的延迟，在大尺度的统计上没有问题，秒级的统计波动却很明显；
- 堆太小会导致worker被强制杀死，所以要配置好-Xmx参数；

自定义项

- 静态资源：静态资源过滤选项，通过Content-Type 或后缀筛选特定的静态资源。

- 资源合并：URL合并，比如RESTful的资源，合并后方便展示；
- 维度与指标：通过ANTLR v3做语法、词法分析，完成自定义维度和指标，并且后续的报警也支持自定义表达式。

其他

我们还用其他方式实现了：

- 业务的进程级(CPU/MEM/端口)监控
- 业务依赖的服务，如MySQL/memcached等的监控
- 服务器的磁盘/内存/IO/内核参数/语言环境/环境变量/编译环境 等监控

原文链接：<http://tech.uc.cn/?p=2866>

专访张路斌：从HTML5到Unity的游戏开发之路

作者：单明珠

摘要：社区之星52期采访了非计算机专业出身、热爱游戏的张路斌，为了离梦想近一些，毕业后前往日本，选择在游戏行业发展。在这段时间里，他在CSDN博客里撰写了几十篇技术文章，并著有《HTML5 Canvas 游戏开发实战》一书。

张路斌，英文名Lufy（博客），非计算机专业出身，由于本身喜欢玩游戏，毕业后千里迢迢前往日本，从事游戏开发工作。一开始接触Java、.Net和PL/SQL开发工作，由于碰上金融危机公司裁员，便跳槽至一家小公司做了半年手机游戏开发，随后到一家互联网公司工作。现在在一家游戏公司上班，接触最多的是Unity开发。

Lufy 曾开发《杨家将传奇》、大型网页游戏《アイドルバトル》、《Flash游戏ポイガチャ》、多平台三国记系列游戏，以及数十款手机小游戏。在CSDN博客上撰写了几十篇的技术博文，还著有《HTML 5 Canvas游戏开发实战》一书，并独立开发了HTML5游戏引擎lufylegend。

近日Lufy接受CSDN社区之星栏目的专访，让我们一起来看看他在日本游戏发展道路上的点点滴滴。

CSDN：请先介绍下自己。

Lufy：大学毕业后，我最先接触Java开发，后来到日本做.Net和PL/SQL开发。很不巧的是，我在日本碰到了严重的经济危机，一起出来的小伙伴们都回国了。相比下，我运气较好，找到了一家做手机游戏开发的小公司，后又跳槽至另一家互联网公司，主要接触PHP、JavaScript和Flash。现在在一家游戏公司工作，接触最多的是Unity。

CSDN: 非计算机专业出身，为什么会选择到日本，在游戏行业发展？

Lufy: 我做这个行业，主要是因为我喜欢玩游戏。游戏玩多了，自然就会有“游戏中的某个地方要是如何如何设计，或许会更好玩”之类的想法，就会想要自己去做一款游戏。

大学时，我做了一款《杨家将传奇》，在同类游戏中，它的人气还算不错，现在也有不少人在玩。这款游戏对我的影响非常大，也更让我坚信游戏开发之路。

毕业后来到日本，很大一部分原因是我比较喜欢日本的游戏，到日本发展或许会让自己离梦想更近。实际上，到去年年末之前，我都不算是一个全职的游戏开发者，我之所以一直在坚持，是因为我很喜欢游戏开发。

HTML5的游戏开发经验之谈——缩短开发周期，并想办法维护

CSDN: 我们知道您曾独立开发大型网页游戏《アイドルバトル》、《Flash游戏ポイガチャ》、多平台三国记系列游戏，以及数十款手机小游戏，能和我们分享下经验吗？

Lufy: 经验谈不上，我就根据自身开发经验简单的说下。之前我开发的有点规模的游戏，现在都已下线了。前几天我听了一个游戏经验的分享，和我的想法不谋而合，我在这里和大家分享下。游戏开发者都知道，一款游戏是否会火，根本就是不可预计的，有的游戏画面特效做得相当绚丽，有的游戏内容非常有意思，有的游戏玩法特别新颖，但最后都被淘汰了。当然，以上这些因素都是一款好游戏应该具备的，但也不是必要的。有时你觉得远不如自己的游戏反而一夜之间火爆了，有些简单的不能再简单的游戏，反而取得了很大成功。

所以，经验告诉我们，游戏开发，就是不断的重复再重复，挑战再挑战，没人知道这个游戏是否会让你或者你的团队“一夜暴富”。

此外，我认为游戏开发应该尽可能的缩短开发周期，让市场来决定你的游戏是否生存下去，然后再想办法维护。就像很多美剧一样，拍摄几集就开始播，先观察观众的反映和需求，反映不好还可以调整，或者直接放弃。当然，还有一些开发者开发游戏是为了自己的兴趣或者单纯的为了实现自己的某个理想，对他们而言，游戏做出来了，就已经算成功了。

CSDN: 2013年时，您写了一本名为《HTML5 Canvas 游戏开发实战》的书，能介绍下吗？

Lufy: 这本书有对HTML5 canvas的API的详细介绍，也有对lufylegend.js引擎的使用详解，更重要的是，书中以实例为向导，详细讲述对休闲、射击、物理以及网络游戏等各种类型游戏的开发流程，包括游戏分析、开发过程、代码解析和小结等相关内容，帮助读者了解每种类型游戏开发的详细步骤，让读者彻底掌握各种类型游戏的开发思想。最后，书中通过数据对比分析，指导读者提升程序的性能、写出高效的代码，从而开发出运行流畅的游戏。

CSDN: 既然您提到了HTML5游戏引擎lufylegend，那么能否介绍下为什么会有自己开发引擎的想法？

Lufy: 至于我为什么想开发自己的HTML5游戏引擎lufylegend，这里我依然引用《HTML5 Canvas 游戏开发实战》一书前言中的一段话来回答我开发HTML5引擎的原由：

我是一个喜欢不断学习新知识的人，所以当HTML5作为一种新技术出现的时候，我没有理由不去了解它。由于本身对JavaScript有一定的了解，所以我在学习HTML5的Canvas时，上手非常快。出于对ActionScript的喜爱，我一开始便试着在JavaScript中模仿ActionScript的API来做开发，并且在博客上发表了《用仿ActionScript的语法来编写HTML5》系列文章，这便是最初的lufylegend开源库件的构建过程。当我把自己研究的类库整合到一起后，发现它使用起来十分方便，使用它来开发游戏可以节约大量的开发时间，于是我将其分享到了网上供大家免费使用，希望给相关开发者提供便利。

CSDN: lufylegend有哪些优势呢？

Lufy: lufylegend 的优势在于入门简单、性能高等特点。其实所有的引擎都有一套自己的标准，并在这个标准上进行封装和扩展，所以在渲染过程中必然要增加很多额外的处理和计算等，但这些都会导致引擎效率的降低。而我在这款引擎的设计和維護上，一直坚持以高性能为第一目标，尽量简化渲染流程，以达到接近原生渲染的速度。

我之前做过一个测试发现，在Canvas 2D基础上，lufylegend的渲染速度高出其他引擎很大一截。目前，lufylegend正在追加WebGL渲染功能，相信不久后的2.0版本，lufylegend在渲染速度上依然会保持领先。

当然一款引擎只比性能是不够的，还要比易用性。在lufylegend交流群里，很多人都说，lufylegend太简单了，用它一天就可以开发出一款简单的小游戏。这个绝不是吹牛，lufylegend在设计上模仿了Flash的API。此外，在lufylegend中还有显示列表、对象、继承、事件等，极大的弥补了JavaScript在开发过程中的不足。

lufylegend中还提供了对Box 2D的简易封装，以及Tween，不同屏幕的自动适配等功能。此外，我还引入了一些在Unity开发中自己发现的一些比较实用的小功能，这都让lufylegend更方便使用。

CSDN：HTML5浏览器兼容性问题让人很头疼，你怎么看待这样问题？

Lufy：说到兼容性，这也是出现许多引擎的原因之一。不同浏览器会有不同的处理，比如不同屏幕大小的自动适配，比如各个浏览器对音频的支持度等。开发者要么自己进行处理，要么就接触第三方工具或者引擎来处理。一款引擎，只有帮助开发者解决问题，才能受到欢迎。

我觉得大家可以对兼容性持乐观态度，因为，兼容性的问题不可能完全消失，但随着一系列标准的完善，这类兼容性问题会越来越小，未来会更小。所以，兼容性、渲染性等问题，应该交给引擎和框架来解决，开发者应该把重心放在自己的产品和开发上。

CSDN：你觉得HTML5在开发游戏时有哪些优势？对它未来发展有哪些看法？

Lufy：用HTML5开发游戏最大优势在于它的跨平台性，即无需进行下载就可进入游戏。一个链接一个二维码就可以在任何平台上向其他人分享你的游戏，还有比这个更简单的传播方式吗？再一个开发JavaScript人员储备充足，这也是一个很大的优势。

HTML5出现的时候，我认为它是未来Web的方向。在移动开发方面，HTML5已经是主流了，这个不用多说。随着移动端和PC端对WebGL等新功能的支持，也让HTML5有了更大的发展空间，我觉得不光是在游戏领域，未来HTML5一定会渗入到各个领域。

Unity能够缩短游戏开发周期，但学习成本高

CSDN：您最近刚换了工作，现任工作最多接触的是Unity开发，可以说您现在也是一位Unity初学者，请问在学习Unity时，遇到了哪些难题？

Lufy：我本身英语比较差，unity的界面是全英文的，所以遇到第一个问题就是打开unity后，眼睛看到的基本都是问号。这个难题我只能自己去查资料、摸索，慢慢学习资料查多了，再多的问号也就变成了文字。

我比较喜欢Flash开发，对于Flash的设计理念根深蒂固，所以刚接触Unity时，遇到2D界面的开发，我总是将Flash的思路带入到Unity中，不过经过公司Unity大牛的指点，最终回归正途。此外，Unity还有自己的一套标准，如果只是将以前完全不同领域的思路或做法强加到Unity当中，只会让后期开发变得越来越困难，这也是导致很多Unity开发者失败的原因之一。

再一个就是unity太复杂，并不是短时间内就可以掌握的，我接触时间还比较短，现在依然在逐步深入学习当中。

CSDN：Unity在3D引擎方面具备卓越的品质和优势，同时也支持2D游戏的开发，您觉得它和HTML5相比，有哪些不同和优劣势？

Lufy：其实Unity和HTML5基本没有冲突点，Unity主要是App开发，而HTML5的优势主要是页游开发或者是依赖于WebView的端游开发，这要看公司的产品侧重哪一块了。

不过既然问到了，我简单说一下自己对Unity的看法。Unity的优点很多，简单总结的话，主要有以下几个方面：

- 相对于游戏引擎来说，功能非常完善；
- 学习资料丰富，交流社区也很强大，开发案例多；
- 可以在PC端预览，Debug方便；
- Editor的扩展方便；
- GUI、以及NGUI等UI组件丰富；
- 多平台支持；
- 可以直接在AssetStore中购买所需素材或组件等。

因为以上优点，使用Unity开发，能够有效的缩短游戏的开发周期。当然缺点也有，比如说学习成本比较高，想短时间深入了解Unity是不可能的。

CSDN：给我们简单的介绍下日本游戏市场吧？

Lufy：这个问题比较大了，我只能简单的说下我对日本手游的一点了解。

- 日本手游中卡牌游戏居多，游戏一般都采取免费下载、内部收费的形势。
- 日本的手游的发布渠道比较单一，一般只考虑苹果以及谷歌旗下的应用商店就可以了。
- 日本用户消费意识很高，日本人对扭蛋尤其钟爱，其也是日本手游的主要收费方式之一，卡牌类、RPG类、养成类、战略类，无论什么类型，扭蛋几乎无处不在，而且所有人都会大把的往里砸钱。
- 日本手机网速比较快，而且手机上网基本上都是包月形势，所以不用担心游戏流量问题。
- 日本人对手机游戏的狂热程度绝对超出你的想象，路上、电车上、厕所里，任何地方都能看到低头摆弄手机玩游戏的人。这也决定了，能够适应碎片化时间的游戏会比较卖座。

CSDN：以后会回国发展吗？怎样看待国内游戏市场的发展？

Lufy：这个当然，以后肯定会回到国内发展的。其实我觉得无论国内还是国外，手游开发都将成为未来游戏开发的主流。而且国内有着全世界最大的用户群，很多国外公司都开始进军中国手游市场，把中国当成最大的游戏市场，包括我现在的公司也是。

现在智能手机在国内已经很普遍了，而且性能越来越高，再加上微信等各种平台渠道的推广，所以未来国内的游戏市场也就是手游市场，手游市场必将取代PC游戏市场。

CSDN：给同样热爱编程游戏的小伙伴们提供一些学习建议吧。

Lufy：这是一个老生常谈的问题，之前也有很多人总结过了，我再总结一遍吧。

- 自己多动手，有些东西看一百遍或者听一百遍，也不如自己写一遍理解的透彻。
- 多看代码，现在开源的代码库这么多，这绝对是提高自己编程能力的一个捷径。
- 多跟人交流，有些问题可能自己通过调查解决了，但如果听下其他人的想法，或许会学到更多。
- 尤其在你刚接触到某个新领域的时候，一定要多看书，这个书包括电子书，或者互联网上一些从基础到深入的连载文章。

在开发过程中，最忌讳的就是遇到问题不思考就发问，虽然我觉得大家都知道这样不好，但是这类人确实有很多。举个简单的例子，一个对象的某个属性可以设定为两个不同的值，对于会学习的人来说，他会将这两个值分别设定，然后看一下结果有什么不同。而另一部分人，会直接到论坛或者QQ群等地方去问。这就是自学能力差异的体现。

原文链接：<http://www.csdn.net/article/2014-08-25/2821359>

Lisp魔咒：对Lisp的非技术性吐槽

作者：Rudolf Winestock

译者：Mort Yao

这篇文章是另一次尝试，旨在解释Lisp语言在具备强大力量的同时、为何Lisp社区却无法重现它们在“AI之冬”之前的辉煌。毫无疑问，即便在式微之后，Lisp语言仍然是各种奇思妙想的重要来源。这一事实，加之各种Lisp实现的优异架构，还有如今在长达十年之后Lisp的复兴、显示着那些Lisp拥护者们是多么需要为自己的得意之作找到一点优越感；尽管有这一切，他们却没能成功地把Lisp的力量转换成一 场压倒性的技术革新。

在本文中，我将阐述这样一个论点：Lisp那极其强大的表达能力，实际上是如何成为它缺乏前进动力的致命诱因的。

Lisp的力量也是它自身最危险的天敌。

要证明这件事情，试想一个简单的思维实验：选择两种非面向对象的程序语言。你的任务，如果你愿意接受的话，就是为它们提供面向对象编程范式的支持，并且保持它们与原语言向后兼容——排除一些边界情况以外。把任意两种语言放到这个思维实验的设定当中，你会很容易发现一些语言较另一些语言更容易完成任务。这正是这个思维实验的要点。随手举个简单的例子：INTERCAL和Pascal。

现在让我们把这个思维实验变得更有说服力些。想象一下给C和Scheme添加面向对象的支持。让Scheme支持面向对象不过是个稍微费点脑筋的家庭作业。另一方面，让C支持面向对象，你恐怕得有Bjarne Stroustrup的本事才能办到。

这种对于解决问题所需才能和努力程度上的分歧，造成了Lisp的魔咒：

Lisp是如此强大，以至于在其他编程语言中的技术问题，到了Lisp中变成了社会问题。

想一想Scheme的情形吧。因为让Scheme支持面向对象是如此轻而易举，许多Scheme黑客都完成过这件事情，更准确地说，是许多独立的Scheme黑客都完成过。这就导致了在20世纪90年代，这个语言的面向对象支持包像工厂量产出来的库存清单一样数不胜数。想想选择谬论就知道，没有哪一个包能够成为正式的标准。如今某些Scheme实现已经有了它们自己的面向对象功能，这还不算太坏。尽管如此，那些五花八门的由不同独立开发者开发出来的包所造成的问题，正如Olin Shivers在给Scheme Shell (scsh) 写文档的时候所提到的一样。

独立黑客们写出来的程序基本上遵循“抓痒模型”。这些程序解决了写程序的黑客们自己关心的问题，但是却未必能很好地处理对于其他人来说有用的部分功能。况且，虽说这些程序无疑可以在这个黑客自己的环境配置上运行得很好，但却不一定能移植到其他的Scheme实现、甚至不同平台上的同一Scheme实现上。文档可能会极度匮乏。从现实的角度讲，一个黑客用自己挤出来的空闲时间做出来的项目，当然也可能会因为黑客自己的现实生活问题而烂尾。正如Olin Shivers指出的那样，这些个人性质的项目更倾向于解决整个问题的百分之八十。

Mark Tarver博士的文章，The Bipolar Lisp Programmer（双面Lisp程序员），对这种现象有一个确切的表述。他写道，这些“孤狼”式的Lisp黑客以及他们：

.....不能把事情恰当地做完收尾。所谓的“用过就扔设计”绝对和拉屎这件事儿没什么两样（注：原文如此），而它来源于Lisp社区。Lisp允许你如此虎头蛇尾地了结一件事，而且还让你心安理得地这么去做。十年前，我有一次需要给我的Lisp程序写一个GUI。没问题，我找到了9个GUI库。麻烦在于这9个库没有一个拥有足够完整的文档，而且它们全部是bug充斥的。基本上，每个人都实现了他们自己的一套解决方案，对于他们来说能用就行。这是一种拉屎式的态度（注：原文如此）；反正我做到了，我消化它了。这同样也是无须在他人帮助下即可得到的产物。

那么再想一想C语言在上述思维实验中的情形吧。由于在C上面实现面向对象支持的困难程度，只有两个严肃的解决方案摆上了台面：C++，以及Objective-C。Objective-C在Mac上最为流行，而C++几乎统治了其他一切平台。这意味着，给定一个平台，如何让C支持面向对象的扩展几乎已经被唯一确定了。这意味着这些语言的面向对象支持将拥有完善的文档，高度集成的IDE，和兼容性良好的库，等等。

Mark Tarver博士的文章说到这一点：

现在与之相反，C/C++的做事方式完全不同。用镊子和胶水一步步搭建成一个东西实在太他妈困难了，所以你所做的一切都是实实在在的成就。你想要为它好好地写些文档。你会在做一个规模可观的C语言项目时候需要他人的帮助；因此你也更加容易变得社会性、学会去与他人合作。你需要做到这些，因为你需要完成某件事情。

而全部的这些，从一个雇主的角度来讲，是非常吸引人的。十个能够相互交流、写文档与团队协作的人显然会比一个像拉翔一样自己去hack些Lisp代码的人更有用，而这种翔的替代品只能是另一坨翔，这些翔们随时都可能因为某些个人的问题、自己退出项目而丢下一个不可收拾的烂摊子。

所以说，那些懂C的人不会去纠结“我应该用哪种面向对象系统？”相反，他们会去选择C++或者Objective-C，就像他们的同事所选择的一样，然后他们会提问“我该怎样使用面向对象的功能X？”答案很简单：“咕狗一下，你就知道。”

真正的黑客，当然早就知道面向对象并非如它的拥趸们所宣称的那样是解决一切问题的灵丹妙药。真正的黑客已经在探寻更加高阶的概念，诸如不可变数据结构、类型推断、惰性求值、monad、arrow、模式匹配、约束编程，等等。真正的黑客也都知道，C/C++对于写大部分不需要进行任意位操作的程序来说并不合适。尽管如此，Lisp的魔咒仍然存在。

一些沾沾自喜的Lisp发烧友，调研了当前学术界编程语言的硕果（Haskell、OCaml等等）后，发现它们所缺失的一些特性，要么就是已经在Lisp中存在、要么就是可以用Lisp很轻易地实现——并且可以改进——基于Lisp宏。他们也许是对的。

但是太可惜了，Lisp黑客们。

Mark Tarver博士——在上面已经两次引用过——曾经设计过一个Lisp的方言，叫做Qi。它仅仅由少于一万行的宏组成，基于Clisp运行。它实现了绝大部分Haskell和OCaml所独有的特性。从某个方面来说，Qi甚至超越了它们。举个例子说吧，Qi的类型推断引擎本身是图灵完全的。在这样的一个由天才科学家组成的杰出团队才能开发出Haskell语言的世界中，Tarver博士，他完全是一个人做出来了Qi。

再看一眼上面这段话。仔细想想，是不是可怕极了。

给读者的习题：假想Haskell与Common Lisp之间发生了激烈的对抗，下一步将会发生什么？

答案：Lisp魔咒应验了。每两个或者三个Lisp黑客就会开发出一套属于自己的惰性求值、纯函数式、`arrow`、模式匹配、类型推断等等的实现。大部分这种项目都是孤狼式开发。因而，它们具备大部人所需要的80%功能（虽然这80%的定义会随情况不同而各异）。它们的文档通常会很差。它们无法在不同的Lisp系统上移植。有些项目可能起初信誓旦旦地会维护下去，直到开发者决定自己跑到别处赚钱去了，结果丢下一个无法收拾的烂摊子。有一些可能会在某种程度上多多少少地打败Haskell，但是它的认可度会被comp.lang.lisp新闻组里面的口水战淹没。

故事的结局：任意一个传统的Lisp黑客的宏能够拼凑成一个文档匮乏的、不可移植的、bug充斥的80%的Haskell实现，仅仅因为Lisp是一种比Haskell表达力更加强大的语言。

这个故事的教育意义在于，次级效应和三级效应至关重要。技术不只是影响我们解决技术问题的方式，它也影响着我们的社会行为。而社会行为会反馈并施加影响于我们最初试图解决的技术问题。

Lisp是一个活生生的事例。Lisp是如此强大有力，它鼓励个人的、狂热的特立独行。在Lisp机器曾经盛极一时的年代，这种特立独行产生了举世瞩目

的成果。但也正是同样的特立独行，阻碍了所谓“自底而上纯Lisp实现”的计算机系统的复苏；自Symbolics和LMI夭折之后，再也没有一个“Lisp操作系统”项目达到过值得令人关注的高度。

这些次级和三级效应的一个后果是，即使Lisp是有史以来最富于表达力的编程语言，以至于理论上几乎不可能创造出比它更具表达力的语言，Lisper们将仍然有许多从其他编程语言那里学习的东西。Smalltalk程序员们教会了每个人——包括那些Lisp黑客们——多多少少一点关于面向对象的概念。Clean语言和Mozart/Oz也有着一些自己的奇特之处。

Lisp魔咒并不违背Stanislav Datskovskiy的至理名言：雇主们更喜欢可以被取代的雇员，而不是生产率最高的雇员。说得太实在了。你早该醒悟到那些管理学课程只是骗钱的把戏。然而，他这篇文章的最后几行似乎存在问题。请看：

在“自由软件”的世界里，工业界教条仅仅是在口头上被激烈地批判，但却从未在实践中被反对过。那些“办公隔间地狱”里被唯恐避之不及的概念，同样也未曾在业余爱好者中间得到过青睐。

在脚注中，他将Linux作为一个拒绝追求新奇想法的实例。为了例证，他将操作系统作为自己的一个论点（下面评论的1L是SB）。他并没有提到编程语言。Python和Ruby都受到了Lisp的影响。很多它们的饭表示了对Lisp的尊重，而他们的一些兴趣则促进了Lisp的复兴。公正地讲，JavaScript也曾被描述为“披着C外衣的Scheme”，尽管它诞生在那些办公隔间地狱中间。

即便有如此大的影响力，在工业和开源界里，Lisp也仅仅只吸引了一部分程序员的一部分注意力，而这也是拜最近脚本语言的兴起所赐。那些拿着MBA学位的高富帅码农们的思维封闭并不是唯一的原因。“Lisp魔咒”本身能解释更多的事情。

提供给Lisp的、可用的自由开发环境可以作为“Lisp魔咒”的一个例证。

尽管说出来让人难堪，但必须得有人去做这件事情。忘掉Lisp机器吧；我们甚至还没有一个能达到算得上Smalltalk黑客小康标准的开发环境（“我

总赶脚到Lisp是一个牛逼的语言，而Smalltalk是一个牛逼的环境。”——Ramon Leon如是说）。除非他们愿意付上千刀美元，否则Lisp黑客们仍然会受制于Emacs。（翻译君：比肾还贵的LispWorks，包邮哦亲，欲购从速哦）

James Gosling，第一个在Unix上运行的Emacs的作者，恰当地指出了Emacs已经长达二十年没有任何基础上的变动。这是因为，Emacs维护者们只是不断地在这个当年由一个MIT的AI实验室的研究生做的设计上垒代码，那时 Emacs项目仍然间接地从国债那里获得资助。也许Slashdot喷子会反驳说Emacs已经无所不能，它可以完成其它任何开发环境所能做的事情，而且只会完成得更好。不然那些当年曾经用过Lisp机器的也要这么说。

那么，为什么Lisp黑客们没有把那些Smalltalk家伙们给彻底打败呢？为什么他们没有做出一个自由的开发环境，可以从某种程度上唤回Lisp机器曾经的辉煌，即使他们不能够重现出另一个Lisp机器？

这件事没有发生的原因来自于Lisp魔咒。大量的Lisp黑客应该去协作。说得更详细些：大量成为Lisp黑客的人们应该去协作。而且他们应该学会合作去做一个新的设计、而非遵从一个从一开始就写死了的现有设计。这过程中不应该有任何来自外界的压力，例如风险资本家或者企业雇主，来干涉他们做事的方式。

每个项目都会存在参与者的分歧，诸如意见不合、风格或哲学上的冲突。如果这些社会性的问题持续下去，任何大的项目都无法完成，这就产生了一个让它们 倾向于解决的反作用力。“要么我们团结一心，要么我们都会被吊死在同一棵树上”。而Lisp的强大表达能力削弱了这个反作用力；一个人总可以着手去自搞一套。这样，刚愎自用的黑客们认为不值得去应付观点分歧带来的麻烦；因此他们要么退出了合作项目，要么就干脆不参加已有的项目、而选择自力更生从头开始。这就是Lisp魔咒。

我们甚至可以自己去hack Emacs，为了追求个人理念中所谓的“足够好”。于是乎，Lisp诅咒差不多就变成了“坏即是好（Worse is Better）”的同义词。（翻译君：这最后一段有点不知所云。。。）

原译文链接：<http://www.soimort.org/posts/124/>

原文链接：http://www.winestockwebdesign.com/Essays/Lisp_Curse.html